## SQL statements

- Common elements in SQL syntax ............................................. 421  
- Syntax conventions ................................................................. 422  
- Statement applicability indicators .......................................... 424  
- SQL statements ................................................................. 424

## Tables

- System tables ................................................................. 1055  
- Diagnostic tracing tables .................................................. 1069  
- Other tables ................................................................. 1083

## System procedures

- Viewing details about system procedures and functions .......... 1085  
- Web services system procedures ........................................ 1085  
- Roles and privileges system procedures .............................. 1086  
- MAPI and SMTP system procedures .................................... 1086  
- Directory and file system procedures ................................. 1088  
- Secure feature system procedures ....................................... 1089  
- Adaptive Server Enterprise system and catalog procedures ..... 1089  
- Alphabetical list of system procedures ................................. 1092

## Views

- System views ................................................................. 1347  
- Consolidated views ........................................................... 1419  
- Compatibility views .......................................................... 1438  
- Views for Transact-SQL compatibility ................................. 1449

## Index

- Index ............................................................................. 1451
About this book

This book provides reference information for system procedures, and the catalog (system tables and views). It also provides an explanation of the SQL Anywhere implementation of the SQL language (search conditions, syntax, data types, and functions).
SQL language elements

Keywords

Each SQL statement contains one or more keywords. SQL is case insensitive to keywords, but throughout the documentation, keywords are indicated in uppercase.

For example, in the following statement, SELECT and FROM are keywords:

```
SELECT *
FROM Employees;
```

The following statements are equivalent to the one above:

```
Select *
From Employees;
select * from Employees;
sSELECT * FRoM Employees;
```

Some keywords cannot be used as identifiers without surrounding them in double quotes. These are called reserved words. Other keywords, such as DBA, do not require double quotes, and are not reserved words.

Reserved words

Some keywords in SQL are also reserved words. To use a reserved word in a SQL statement as an identifier, enclose it in double quotes, square brackets or back quotes. Many of the keywords that appear in SQL statements are reserved words. For example, use the following syntax to retrieve the contents of a table named SELECT.

```
SELECT *
FROM "SELECT"
```

SQL keywords are not case sensitive and the following words may appear in uppercase, lowercase, or any combination of the two. All strings that differ only in capitalization from one of the following words are reserved words.

You can also turn off keywords using the non_keywords option.

The reserved_keywords option turns on individual keywords that are disabled by default.

If you are using embedded SQL, you can use the sql_needs_quotes database library function to determine whether a string requires quotation marks. A string requires quotes if it is a reserved word or if it contains a character not ordinarily allowed in an identifier.

You can obtain a list of the reserved words using the sa_reserved_words system procedure.

The reserved SQL keywords in SQL Anywhere are:
<table>
<thead>
<tr>
<th>add</th>
<th>all</th>
<th>alter</th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td>any</td>
<td>array</td>
<td>as</td>
<td>asc</td>
</tr>
<tr>
<td>attach</td>
<td>backup</td>
<td>begin</td>
<td>between</td>
</tr>
<tr>
<td>bigint</td>
<td>binary</td>
<td>bit</td>
<td>bottom</td>
</tr>
<tr>
<td>break</td>
<td>by</td>
<td>call</td>
<td>capability</td>
</tr>
<tr>
<td>cascade</td>
<td>case</td>
<td>cast</td>
<td>char</td>
</tr>
<tr>
<td>char_convert</td>
<td>character</td>
<td>check</td>
<td>checkpoint</td>
</tr>
<tr>
<td>close</td>
<td>comment</td>
<td>commit</td>
<td>compressed</td>
</tr>
<tr>
<td>conflict</td>
<td>connect</td>
<td>constraint</td>
<td>contains</td>
</tr>
<tr>
<td>continue</td>
<td>convert</td>
<td>create</td>
<td>cross</td>
</tr>
<tr>
<td>cube</td>
<td>current</td>
<td>current_timestamp</td>
<td>current_user</td>
</tr>
<tr>
<td>cursor</td>
<td>date</td>
<td>datetimeoffset</td>
<td>dbspace</td>
</tr>
<tr>
<td>deallocate</td>
<td>dec</td>
<td>decimal</td>
<td>declare</td>
</tr>
<tr>
<td>default</td>
<td>delete</td>
<td>deleting</td>
<td>desc</td>
</tr>
<tr>
<td>detach</td>
<td>distinct</td>
<td>do</td>
<td>double</td>
</tr>
<tr>
<td>drop</td>
<td>dynamic</td>
<td>else</td>
<td>elseif</td>
</tr>
<tr>
<td>encrypted</td>
<td>end</td>
<td>endif</td>
<td>escape</td>
</tr>
<tr>
<td>except</td>
<td>exception</td>
<td>exec</td>
<td>execute</td>
</tr>
<tr>
<td>existing</td>
<td>exists</td>
<td>externlogin</td>
<td>fetch</td>
</tr>
<tr>
<td>first</td>
<td>float</td>
<td>for</td>
<td>force</td>
</tr>
<tr>
<td>foreign</td>
<td>forward</td>
<td>from</td>
<td>full</td>
</tr>
<tr>
<td>goto</td>
<td>grant</td>
<td>group</td>
<td>having</td>
</tr>
<tr>
<td>holdlock</td>
<td>identified</td>
<td>if</td>
<td>in</td>
</tr>
<tr>
<td>index</td>
<td>inner</td>
<td>inout</td>
<td>insensitive</td>
</tr>
<tr>
<td>insert</td>
<td>inserting</td>
<td>install</td>
<td>instead</td>
</tr>
<tr>
<td>int</td>
<td>integer</td>
<td>integrated</td>
<td>intersect</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>into</td>
<td>is</td>
<td>isolation</td>
<td>join</td>
</tr>
<tr>
<td>json</td>
<td>kerberos</td>
<td>key</td>
<td>lateral</td>
</tr>
<tr>
<td>left</td>
<td>like</td>
<td>limit</td>
<td>lock</td>
</tr>
<tr>
<td>login</td>
<td>long</td>
<td>match</td>
<td>membership</td>
</tr>
<tr>
<td>merge</td>
<td>message</td>
<td>mode</td>
<td>modify</td>
</tr>
<tr>
<td>natural</td>
<td>nchar</td>
<td>new</td>
<td>no</td>
</tr>
<tr>
<td>noholdlock</td>
<td>not</td>
<td>notify</td>
<td>null</td>
</tr>
<tr>
<td>numeric</td>
<td>nvarchar</td>
<td>of</td>
<td>off</td>
</tr>
<tr>
<td>on</td>
<td>open</td>
<td>openstring</td>
<td>openxml</td>
</tr>
<tr>
<td>option</td>
<td>options</td>
<td>or</td>
<td>order</td>
</tr>
<tr>
<td>others</td>
<td>out</td>
<td>outer</td>
<td>over</td>
</tr>
<tr>
<td>passthrough</td>
<td>precision</td>
<td>prepare</td>
<td>primary</td>
</tr>
<tr>
<td>print</td>
<td>privileges</td>
<td>proc</td>
<td>procedure</td>
</tr>
<tr>
<td>publication</td>
<td>raiserror</td>
<td>readtext</td>
<td>real</td>
</tr>
<tr>
<td>reference</td>
<td>references</td>
<td>refresh</td>
<td>release</td>
</tr>
<tr>
<td>remote</td>
<td>remove</td>
<td>rename</td>
<td>reorganize</td>
</tr>
<tr>
<td>resource</td>
<td>restore</td>
<td>restrict</td>
<td>return</td>
</tr>
<tr>
<td>revoke</td>
<td>right</td>
<td>rollback</td>
<td>rollup</td>
</tr>
<tr>
<td>row</td>
<td>rowtype</td>
<td>save</td>
<td>savepoint</td>
</tr>
<tr>
<td>scroll</td>
<td>select</td>
<td>sensitive</td>
<td>session</td>
</tr>
<tr>
<td>set</td>
<td>setuser</td>
<td>share</td>
<td>smallint</td>
</tr>
<tr>
<td>some</td>
<td>spatial</td>
<td>sqlda</td>
<td>sqlstate</td>
</tr>
<tr>
<td>start</td>
<td>stop</td>
<td>subtrans</td>
<td>subtransaction</td>
</tr>
<tr>
<td>synchronize</td>
<td>table</td>
<td>temporary</td>
<td>then</td>
</tr>
</tbody>
</table>
Identifiers are the names of objects in the database, such as user IDs, tables, and columns.

Remarks

Identifiers have a maximum length of 128 bytes. Enclose an identifier in double quotes, square brackets, or back quotes (`...`) if any of the following conditions are true:

- The identifier contains spaces.
- The first character of the identifier is not an alphabetic character.
- The identifier is a reserved word.
- The identifier contains characters other than alphabetic characters and digits.

Alphabetic characters include the alphabet, the underscore character ( `_` ), at sign ( `@` ), number sign ( `#` ), and dollar sign ( `$` ). The database collation sequence dictates which characters are considered alphabetic or digit characters.
The following characters are not permitted in identifiers:

- Double quotes
- Control characters (any character less than 0x20)
- Backslashes
- Square brackets
- Back quotes

**Note**

If you are reloading a database that is of an earlier version than 16.0, remove any square brackets or back quotes in identifiers; otherwise, the reload fails.

If the quoted_identifier database option is set to Off, double quotes are used to delimit SQL strings and cannot be used for identifiers. However, you can use square brackets or back quotes to delimit identifiers, regardless of the setting of quoted_identifier. The default setting for the quoted_identifier option is Off for Open Client and jConnect connections; otherwise, the default is On.

- User IDs cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons

- Passwords are case-sensitive and they cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons
  - be longer than 255 bytes in length

**Standards and compatibility**

- **SQL/2008** The ability to create identifiers of up to 128 characters is optional SQL language feature F391 of the SQL/2008 standard.

**See also**

- “Reserved words” on page 1
- “quoted_identifier option” [SQL Anywhere Server - Database Administration]
Examples

The following are all valid identifiers.

- Surname
- "Client Name"
- `Client Name`
- [Surname]
- SomeBigName

Strings

A string is a sequence of characters up to 2 GB in size. A string can occur in SQL:

- as a string literal. A string literal is a sequence of characters enclosed in single quotes (apostrophes). A string literal represents a particular, constant value, and it may contain escape sequences for special characters that cannot be easily typed as characters.

- as the value of a column or variable with a CHAR or NCHAR data type.

- as the result of evaluating an expression.

The length of a string can be measured in two ways:

- **Byte length**  The byte length is the number of bytes in the string.

- **Character length**  The character length is the number of characters in the string, and is based on the character set being used.

For single-byte character sets, such as cp1252, the byte-length and character-length are the same. For multibyte character sets, a string's byte-length is greater than or equal to its character-length.

See also

- “String literals” on page 8

Constants

This section describes binary literals and string literals.

Binary literals

A binary literal is a sequence of hexadecimal characters consisting of digits 0-9 and uppercase and lowercase letters A-F. When you enter binary data as literals, you must precede the data by 0x (a zero, followed by an x), and there should be an even number of digits to the right of this prefix. For example, the hexadecimal equivalent of 39 is 0027, and is expressed as 0x0027.
Hexadecimal constants in the form of 0x12345678 are treated as binary strings. An unlimited number of digits can be added after the 0x.

A binary literal is sometimes referred to as a binary constant. In SQL Anywhere, the preferred term is binary literal.

**Converting to and from hexadecimal values**

You can use the CAST, CONVERT, HEXTOINT, and INTTOHEX functions to convert a binary string to an integer. The CAST and CONVERT functions convert hexadecimal constants to TINYINT, signed and unsigned 32-bit integer, signed and unsigned 64-bit integer, NUMERIC, and so on. The HEXTOINT function only converts a hexadecimal constant to a signed 32-bit-integer.

The value returned by the CAST function cannot exceed 8 digits. Values exceeding 8 digits return an error. Zeros are added to the left of values less than 8 digits. For example, the following argument returns the value -2,147,483,647:

```sql
SELECT CAST ( 0x0080000001 AS INT );
```

The following argument returns an error because the 10-digit value cannot be represented as a signed 32-bit integer:

```sql
SELECT CAST ( 0xff80000001 AS INT );
```

The value returned by the HEXTOINT function can exceed 8 digits if the value can be represented as a signed 32-bit integer. The HEXTOINT function accepts string literals or variables consisting only of digits and the uppercase or lowercase letters A-F, with or without a 0x prefix. The hexadecimal value represents a negative integer when the 8th digit from the right is one of the digits 8-9, the uppercase or lowercase letters A-F, or the previous leading digits are all uppercase or lowercase letter F.

The following arguments return the value -2,147,483,647:

```sql
SELECT HEXTOINT( '0xFF80000001' );
SELECT HEXTOINT( '0x80000001' );
SELECT HEXTOINT ( '0xFFFFFFFFFFFFFFFF80000001' );
```

The following argument returns an error because the argument represents a positive integer value that cannot be represented as a signed 32-bit integer:

```sql
SELECT HEXTOINT( '0x0080000001' );
```

**See also**

- “CAST function [Data type conversion]” on page 174
- “CONVERT function [Data type conversion]” on page 187
- “HEXTOINT function [Data type conversion]” on page 258
- “INTTOHEX function [Data type conversion]” on page 274
String literals
A string literal is a sequence of characters enclosed in single quotes. For example, 'Hello world' is a string literal of type CHAR. Its byte length is 11, and its character length is also 11.

A string literal is sometimes referred to as a string constant, literal string, or just as a string. In SQL Anywhere, the preferred term is string literal.

You can specify an NCHAR string literal by prefixing the quoted value with N. For example, N'Hello world' is a string literal of type NCHAR. Its byte length is 11, and its character length is 11. The bytes within an NCHAR string literal are interpreted using the database's CHAR character set, and then converted to NCHAR. The syntax N' string' is a shortened form for CAST( ' string' AS NCHAR ).

Escape sequences
Sometimes you must put characters into string literals that cannot be typed or entered normally. Examples include control characters (such as a new line character), single quotes (which would otherwise mark the end of the string literal), and hexadecimal byte values. For this purpose, you use an escape sequence.

The following examples show how to use escape sequences in string literals.

- A single quote is used to mark the beginning and end of a string literal, so a single quote in a string must be escaped using an additional single quote, as follows: 'John''s database'
- A backslash followed by any character other than n, x, X, or \ is interpreted as two separate characters. For example, \q inserts a backslash and the letter q.
  Hexadecimal escape sequences can be used for any character or binary value. A hexadecimal escape sequence is a backslash followed by an x followed by two hexadecimal digits. The hexadecimal value is interpreted as a character in the CHAR character set for both CHAR and NCHAR string literals. The value \x09 must be coded as \x09 if you don't want the value stored as a single tab character, but \xyy would be stored as \xyy. The following example, in code page 1252, represents the digits 1, 2, and 3, followed by the euro currency symbol: '123\x80'.
- Escape a backslash character by using an additional backslash, as follows: 'c:\\november'. For paths, you can also use the forward slash (/) instead of a backslash: 'c:/november'.
- Represent a new line character by using a backslash followed by n (\n), specify: 'First line: \nSecond line:'

You can use the same characters and escape sequences with NCHAR string literals as with CHAR string literals.

To use Unicode characters that cannot be typed directly into the string literal, use the UNISTR function.

See also
- “UNISTR function [String]” on page 398
Operators

This section describes arithmetic, string, array, and bitwise operators.

The normal precedence of operations applies. Expressions in parentheses are evaluated first, then multiplication and division before addition and subtraction. String concatenation happens after addition and subtraction.

See also

- “Operator precedence” on page 20
- “Search conditions” on page 40

Comparison operators

The syntax for comparison is as follows:

(expression comparison-operator expression)

where comparison-operator is one of the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>!&gt;</td>
<td>Not greater than</td>
</tr>
<tr>
<td>!&lt;</td>
<td>Not less than</td>
</tr>
</tbody>
</table>

Case sensitivity

By default, SQL Anywhere databases are created as case insensitive. Comparisons are carried out with the same attention to case as the database they are operating on. You can control the case sensitivity of SQL Anywhere databases with the -c option when you create the database.

Case sensitivity is set at database creation.
Note
All string comparisons are case insensitive unless the database was created as case sensitive.

Trailing blanks
The behavior of SQL Anywhere when comparing strings is set at database creation.

See also
- “Case sensitivity” [SQL Anywhere Server - SQL Usage]
- “Ignore trailing blanks in comparisons” [SQL Anywhere Server - SQL Usage]

Logical operators
Search conditions can be combined using the AND or OR operators. You can also negate them using the NOT operator, or test whether an expression would evaluate to true, false, or unknown, using the IS operator.

- **AND operator** The AND operator is placed between search conditions as follows:

  ...WHERE condition1 AND condition2

  When using AND, the combined condition is TRUE if both conditions are TRUE, FALSE if either condition is FALSE, and UNKNOWN otherwise.

- **OR operator** The OR operator is placed between search conditions as follows:

  ...WHERE condition1 OR condition2

  When using OR, the combined condition is TRUE if either condition is TRUE, FALSE if both conditions are FALSE, and UNKNOWN otherwise.

- **NOT operator** The NOT operator is placed before a condition to negate the condition, as follows:

  ...WHERE NOT condition

  The NOT condition is TRUE if condition is FALSE, FALSE if condition is TRUE, and UNKNOWN if condition is UNKNOWN.

- **IS operator** The IS operator is placed between an expression and the truth value you are testing for. The syntax for the IS operator is as follows:

  expression IS [ NOT ] truth-value

  The IS condition is TRUE if the expression evaluates to the supplied truth-value, which must be one of TRUE, FALSE, UNKNOWN, or NULL. Otherwise, the value is FALSE.

  For example, 5*3=15 IS TRUE tests whether the expression 5*3=15 evaluates to TRUE.
Arithmetic operators

- **expression + expression**  
  Addition. If either expression is the NULL value, the result is NULL.

- **expression - expression**  
  Subtraction. If either expression is the NULL value, the result is NULL.

- **-expression**  
  Negation. If the expression is the NULL value, the result is NULL.

- **expression * expression**  
  Multiplication. If either expression is NULL, the result is NULL.

- **expression / expression**  
  Division. If either expression is NULL or if the second expression is 0, the result is NULL.

- **expression % expression**  
  Modulo finds the integer remainder after a division involving two whole numbers. For example, 21 % 11 = 10 because 21 divided by 11 equals 1 with a remainder of 10.

Standards and compatibility

- **SQL/2008**  
  The use of % as a modulus operator is a vendor extension.

String operators

- **expression || expression**  
  String concatenation (two vertical bars). If either string is NULL, it is treated as the empty string for concatenation.

- **expression + expression**  
  Alternative string concatenation. When using the + concatenation operator, you must ensure the operands are explicitly set to character data types rather than relying on implicit data conversion.

For example, the following query returns the integer value 579:

```sql
SELECT 123 + 456;
```

whereas the following query returns the character string 123456:

```sql
SELECT '123' + '456';
```

You can use the CAST or CONVERT function to explicitly convert data types.

Standards and compatibility

- **SQL/2008**  
  The || operator is the SQL/2008 string concatenation operator. However, in the SQL standard, if either operand of || is the NULL value, then the result of the concatenation is also NULL. With SQL Anywhere, the || operator treats NULL as an empty string.
OPENXML operator

Generates a result set from an XML document.

Syntax 1

\[
\text{OPENXML(} \\
xml-data \\
\quad , xpath \\
\quad [, flags] \\
\quad [, namespaces ] ] \\
\quad \text{WITH ( column-name column-type} \\
\quad \quad [ xpath ] [ , ... ] \\
\text{)}
\]

Syntax 2

\[
\text{OPENXML(} \{ \text{USING FILE} \mid \text{USING VALUE}\} \\
xml-data \\
\quad , xpath \\
\quad [, flags] \\
\quad [, namespaces ] ] \\
\quad \text{WITH ( column-name column-type} \\
\quad \quad [ xpath ] [ , ... ] \\
\quad \text{OPTION ( scan-option ) } \\
\quad \text{AS } \text{correlation-name}
\]

\[
\text{ scan-option } : \\
\text{ ENCODING encoding} \\
\quad \mid \text{BYTE ORDER MARK } \{ \text{ON} \mid \text{OFF} \}
\]

Arguments

- **WITH clause** Specifies the schema of the result set and how the value is found for each column in the result set. WITH clause xpath arguments are matched relative to the matches for the xpath in the second argument. If a WITH clause expression matches more than one node, then only the first node in the document order is used. If the node is not a text node, then the result is found by appending all the text node descendants. If a WITH clause expression does not match any nodes, then the column for that row is NULL.

The xpath arguments in the WITH clause can be literal strings or variables.

The OPENXML WITH clause syntax is similar to the syntax for selecting from a stored procedure.

- **USING FILE | USING VALUE** Use the USING FILE clause to load data from a file.

Note

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].
Use the USING VALUE clause to load data from any expression of CHAR, NCHAR, BINARY, or LONG BINARY type, or BLOB string.

- **xml-data**  
The XML on which the result set is based. This can be any string expression, such as a constant, variable, or column.

  The xml-data is parsed directly in the NCHAR encoding if there are any NCHAR columns in the output. The xpath and namespaces arguments are also converted and parsed in the NCHAR encoding.

- **xpath**  
A string containing an XPath query. XPath allows you to specify patterns that describe the structure of the XML document you are querying. The XPath pattern included in this argument selects the nodes from the XML document. Each node that matches the XPath query in the second xpath argument generates one row in the table.

Metaproperties can only be specified in WITH clause xpath arguments. A metaproperty is accessed within an XPath query as if it was an attribute. If a namespaces is not specified, then by default the prefix mp is bound to the Uniform Resource Identifier (URI) urn:ianywhere-com:saxpath-metaprop. If a namespaces is specified, this URI must be bound to mp or some other prefix to access metaproperties in the query. Metaproperty names are case sensitive. The OPENXML statement supports the following metaproperties:

- **@mp:id** returns an ID for a node that is unique within the XML document. The ID for a given node in a given document may change if the database server is restarted. The value of this metaproperty increases with document order.

- **@mp:localname** returns the local part of the node name, or NULL if the node does not have a name.

- **@mp:prefix** returns the prefix part of the node name, or NULL if the node does not have a name or if the name is not prefixed.

- **@mp:namespaceuri** returns the URI of the namespace that the node belongs to, or NULL if the node is not in a namespace.

- **@mp:xmltext** returns a subtree of the XML document in XML form. For example, when you match an internal node, you can use this metaproperty to return an XML string, rather than the concatenated values of the descendant text nodes.

- **flags** Indicates the mapping that should be used between the XML data and the result set when an XPath query is not specified in the WITH clause. If the flags parameter is not specified, the default behavior is to map attributes to columns in the result set. The flags parameter can have one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XML attributes are mapped to columns in the result set (the default).</td>
</tr>
<tr>
<td>2</td>
<td>XML elements are mapped to columns in the result set.</td>
</tr>
</tbody>
</table>
*namespace-declaration* An XML document. The in-scope namespaces for the query are taken from the root element of the document. If namespaces are specified, then you must include a *flags* argument, even if all the *xpath* arguments are specified.

- **column-name** The name of the column in the result set.

- **column-type** The data type of the column in the result set. The data type must be compatible with the values selected from the XML document.

- **OPTION clause** Use the OPTION clause to specify parsing options to use for the input file, such as escape characters, delimiters, encoding, and so on.

  - **ENCODING clause** The ENCODING clause allows you to specify the encoding that is used to read the file.

    If the ENCODING clause is not specified, then encoding for values is assumed to be in the database character set (db_charset) if the values are of type CHAR or BINARY, and NCHAR database character set (nchar_charset) if the values are of type NCHAR.

  - **BYTE ORDER MARK clause** Use the BYTE ORDER MARK clause to specify whether a byte order mark (BOM) is present in the encoding. By default, this option is ON, which enables the server to search for and interpret a byte order mark (BOM) at the beginning of the data. If BYTE ORDER MARK is OFF, the server does not search for a BOM.

    You must specify the BYTE ORDER MARK clause if the input data is encoded.

    If the ENCODING clause is specified:

      - If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding with an endian such as UTF-16BE or UTF-16LE, the database server searches for a BOM at the beginning of the data. If a BOM is present, it is used to verify the endianness of the data. If you specify the wrong endian, an error is returned.

      - If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding without an explicit endian, the database server searches for a BOM at the beginning of the data. If a BOM is present, it is used to determine the endianness of the data. Otherwise, the operating system endianness is assumed.

      - If the BYTE ORDER MARK option is ON and you specify a UTF-8 encoding, the database server searches for a BOM at the beginning of the data. If a BOM is present it is ignored.

    If the ENCODING clause is not specified:

      - If you do not specify an ENCODING clause and the BYTE ORDER MARK option is ON, the server looks for a BOM at the beginning of the input data. If a BOM is located, the source encoding is automatically selected based on the encoding of the BOM (UTF-16BE, UTF-16LE, or UTF-8) and the BOM is not considered to be part of the data to be loaded.

      - If you do not specify an ENCODING clause and the BYTE ORDER MARK option is OFF, or a BOM is not found at the beginning of the input data, the database CHAR encoding is used.
Remarks

The OPENXML operator parses the xml-data and models the result as a tree. The tree contains a separate node for each element, attribute, and text node, or other XML construct. The XPath queries supplied to the OPENXML operator are used to select nodes from the tree, and the selected nodes are then mapped to the result set.

The XML parser used by the OPENXML operator is non-validating, and does not read the external DTD subset or external parameter entities.

When there are multiple matches for a column expression, the first match in the document order (the order of the original XML document before it was parsed) is used. NULL is returned if there are no matching nodes. When an internal node is selected, the result is all the descendant text nodes of the internal node concatenated together.

Columns of type BINARY, LONG BINARY, IMAGE, and VARBINARY are assumed to be in base64-encoded format and are decoded automatically. If you generate XML using the FOR XML clause, these types are base64-encoded, and can be decoded using the OPENXML operator.

The OPENXML operator supports a subset of the XPath syntax, as follows:

- The child, self, attribute, descendant, descendant-or-self, and parent axes are fully supported.
- Both abbreviated and unabbreviated syntax can be used for all supported features. For example, 'a' is equivalent to 'child::a' and '..' is equivalent to 'parent::node()'.
- Name tests can use wildcards. For example, 'a/*/b'.
- The following kind tests are supported: node(), text(), processing-instruction(), and comment().
- Qualifiers of the form expr1[expr2] and expr1[expr2="string"] can be used, where expr2 is any supported XPath expression. A qualifier evaluates TRUE if expr2 matches one or more nodes. For example, 'a [b]' finds a nodes that have at least one b child, and a [b="I"] finds a nodes that have at least one b child with a text value of I.

Privileges

If the USING FILE clause is specified, you must have the READ FILE system privilege. Otherwise, no privileges are required.

See also

- “FOR XML and binary data” [SQL Anywhere Server - SQL Usage]
- “Using XPath expressions” [SQL Anywhere Server - SQL Usage]
- “SQL data types” on page 89
- “XML import using the OPENXML operator” [SQL Anywhere Server - SQL Usage]
- “Supported character sets” [SQL Anywhere Server - Database Administration]
- “FROM clause” on page 810
- XPath query language: http://www.w3.org/TR/xpath.
Example

The following query generates a result set from the XML document supplied as the first argument to the OPENXML operator:

```sql
SELECT * FROM OPENXML(''<products>
  <ProductType ID="301">Tee Shirt</ProductType>
  <ProductType ID="401">Baseball Cap</ProductType>
</products>'',
'/products/ProductType' )
WITH ( ProductName LONG VARCHAR 'text()', ProductID CHAR(3) '@ID');
```

This query generates the following result:

<table>
<thead>
<tr>
<th>ProductName</th>
<th>ProductID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tee Shirt</td>
<td>301</td>
</tr>
<tr>
<td>Baseball Cap</td>
<td>401</td>
</tr>
</tbody>
</table>

In the following example, the first <ProductType> element contains an entity. When you execute the query, this node is parsed as an element with four children: Tee, &, Sweater, and Set. You can use . to concatenate the children together in the result set.

```sql
SELECT * FROM OPENXML(''<products>
  <ProductType ID="301">Tee & Sweater Set</ProductType>
  <ProductType ID="401">Baseball Cap</ProductType>
</products>'',
'/products/ProductType' )
WITH ( ProductName LONG VARCHAR '.', ProductID CHAR(3) '@ID');
```

This query generates the following result:

<table>
<thead>
<tr>
<th>ProductName</th>
<th>ProductID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tee Shirt &amp; Sweater Set</td>
<td>301</td>
</tr>
<tr>
<td>Baseball Cap</td>
<td>401</td>
</tr>
</tbody>
</table>

The following query uses an equality predicate to generate a result set from the supplied XML document.

```sql
SELECT * FROM OPENXML(''<EmployeeDirectory>
  <Employee>
    <column name="EmployeeID">105</column>
    <column name="GivenName">Matthew</column>
    <column name="Surname">Cobb</column>
    <column name="Street">7 Pleasant Street</column>
    <column name="City">Grimsby</column>
    <column name="State">UT</column>
    <column name="PostalCode">02154</column>
    <column name="Phone">6175553840</column>
  </Employee>
  <Employee>
    <column name="EmployeeID">148</column>
    <column name="GivenName">Julie</column>
    <column name="Surname">Jordan</column>
    <column name="Street">1244 Great Plain Avenue</column>
  </Employee>
</EmployeeDirectory>'',
'/<EmployeeDirectory>'
WITH ( EmployeeID LONG VARCHAR 'text()', GivenName LONG VARCHAR 'text()', Surname LONG VARCHAR 'text()', Street LONG VARCHAR 'text()', City LONG VARCHAR 'text()', State LONG VARCHAR 'text()', PostalCode LONG VARCHAR 'text()', Phone LONG VARCHAR 'text()');
```
<column name="City">Woodbridge</column>
<column name="State">AZ</column>
<column name="PostalCode">01890</column>
<column name="Phone">6175557835</column>
</Employee>

<Employee>
<column name="EmployeeID">160</column>
<column name="GivenName">Robert</column>
<column name="Surname">Breault</column>
<column name="Street">358 Cherry Street</column>
<column name="City">Milton</column>
<column name="State">PA</column>
<column name="PostalCode">02186</column>
<column name="Phone">6175553099</column>
</Employee>

<Employee>
<column name="EmployeeID">243</column>
<column name="GivenName">Natasha</column>
<column name="Surname">Shishov</column>
<column name="Street">151 Milk Street</column>
<column name="City">Grimsby</column>
<column name="State">UT</column>
<column name="PostalCode">02154</column>
<column name="Phone">6175552755</column>
</Employee>

</EmployeeDirectory>

WITH ( EmployeeID INT '@EmployeeID',
  GivenName    CHAR(20) '@GivenName',
  Surname      CHAR(20) '@Surname',
  PhoneNumber  CHAR(10) '@Phone');

This query generates the following result set:

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>GivenName</th>
<th>Surname</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>Matthew</td>
<td>Cobb</td>
<td>6175553840</td>
</tr>
<tr>
<td>148</td>
<td>Julie</td>
<td>Jordan</td>
<td>6175557835</td>
</tr>
<tr>
<td>160</td>
<td>Robert</td>
<td>Breault</td>
<td>6175553099</td>
</tr>
<tr>
<td>243</td>
<td>Natasha</td>
<td>Shishov</td>
<td>6175552755</td>
</tr>
</tbody>
</table>

The following query uses the XPath @attribute expression to generate a result set:

SELECT * FROM OPENXML( 'Employee
  EmployeeID="105"
  GivenName="Matthew"
  Surname="Cobb"
  Street="7 Pleasant Street"
  City="Grimsby"
  State="UT"
  PostalCode="02154"
  Phone="6175553840"
/>', '/Employee' )
WITH ( EmployeeID INT '@EmployeeID',
  GivenName    CHAR(20) '@GivenName',
  Surname      CHAR(20) '@Surname',
  PhoneNumber  CHAR(10) '@Phone');
The following query operates on an XML document like the one used in the above query, except that an XML namespace has been introduced. It demonstrates the use of wildcards in the name test for the XPath query, and generates the same result set as the above query.

```sql
SELECT * FROM OPENXML( '<Employee xmlns="http://www.iAnywhere.com/EmployeeDemo" EmployeeID="105" GivenName="Matthew" Surname="Cobb" Street="7 Pleasant Street" City="Grimsby" State="UT" PostalCode="02154" Phone="6175553840" />' , '/*:Employee' )
WITH ( EmployeeID INT '@EmployeeID', GivenName    CHAR(20) '@GivenName', Surname      CHAR(20) '@Surname', PhoneNumber  CHAR(10) '@Phone');
```

Alternatively, you could specify a namespace declaration:

```sql
WITH ( EmployeeID INT '@EmployeeID', GivenName    CHAR(20) '@GivenName', Surname      CHAR(20) '@Surname', PhoneNumber  CHAR(10) '@Phone');
```

### Array operators

- **expression || expression**  
  Array concatenation (two vertical bars). If either array is NULL, it is treated as a zero-length array for concatenation.

### Standards and compatibility

- **SQL/2008**  
  The || operator is the SQL/2008 concatenation operator. However, in the SQL standard, if either operand of || is the NULL value, then the result of the concatenation is also NULL. With SQL Anywhere, the || operator treats NULL as a zero-length array.

### UNNEST array operator

Creates a derived table from the given array expressions that results in one row per array element.
Syntax

```
UNNEST ( array-expression [, ... ] )
[ WITH ORDINALITY ]
```

Arguments

- `array-expression` An array to derive a table column from.
- `WITH ORDINALITY` The WITH ORDINALITY clause permits the application to recall the original array element from which each value was obtained. Valid UNNEST derived tables must have names specified (by using the AS clause) for each of the resulting expressions. The order of the resulting rows from unnest is not guaranteed. Users can achieve a desired ordering with an ORDER BY clause.

Remarks

If the array expressions have different cardinalities, the missing output expressions from the shorter array(s) are set to NULL. If the WITH ORDINALITY clause is specified, the result set contains an integer column that identifies the array element's cardinal number that the row represents. The new column is appended to the unnest derived table as its last column.

Privileges

None

See also

- “Composite data types” on page 128
- “ARRAY constructor [Composite]” on page 157
- “Comparisons of composite types” on page 136

Example

The following example illustrates how to use of the unnest operator with two arrays that have different cardinalities:

```
SELECT * FROM UNNEST( ARRAY(2,3,4) , ARRAY(4,5,6) ) WITH ORDINALITY AS
    DT(X,Y,Z);
```

The SQL statement returns the following result:

```
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>
```

Bitwise operators

The following operators can be used on bit data types, integer data types (including all variants such as bit, tinyint, smallint and so on), binary values, and bit array data types in SQL Anywhere.
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>bitwise AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>bitwise exclusive OR</td>
</tr>
<tr>
<td>~</td>
<td>bitwise NOT</td>
</tr>
</tbody>
</table>

The bitwise operators &, | and ~ are not interchangeable with the logical operators AND, OR, and NOT.

### Standards and compatibility

- **SQL/2008**  
  Vendor extension. Bitwise operators, along with the BIT VARYING and BIT data types, were supported in the SQL/1999 standard as SQL language feature F511. This feature was eliminated from the SQL/2003 standard.

### Example

The following statement selects rows in which the correct bits are set. For example, if the value of `options` is `0x1001` then the row would be included.

```sql
SELECT *
FROM tableA
WHERE ( options & 0x0101 ) <> 0;
```

### Join operators

#### Note

Support for Transact-SQL outer join operators *= and =* is deprecated. To use Transact-SQL outer joins, the `tsql_outer_joins` database option must be set to `On`.

SQL Anywhere supports two additional comparison operators, *= and =*, which are the Transact-SQL outer join operators. When one of these operators is used in a comparison predicate, an implicit LEFT or RIGHT OUTER JOIN is specified.

### See also

- “`tsql_outer_joins` option” [SQL Anywhere Server - Database Administration]
- “Transact-SQL outer joins (*= or =*)” [SQL Anywhere Server - SQL Usage]

### Operator precedence

The precedence of operators in expressions is as follows. The operators at the top of the list are evaluated before those at the bottom of the list.

1. unary operators (operators that require a single operand)
Expressions

An expression is a statement that can be evaluated to return values.

Syntax

expression:
case-expression
  constant
  [correlation-name.]column-name
  - expression
  expression operator expression
  ( expression )
  function-name ( expression, ... )
  if-expression
  special value
  ( subquery )
  variable-name
  sequence-expression

case-expression:
  CASE expression
  WHEN expression
  THEN expression,...
  [ ELSE expression ]
  END

alternative form of case-expression:
  CASE
  WHEN search-condition
  THEN expression, ...
  [ ELSE expression ]
  END

constant:
  integer | number | string | host-variable
special-value :
  CURRENT { DATE | TIME | TIMESTAMP }
  NULL
  SQLCODE
  SQLSTATE
  USER

if-expression :
  IF condition
  THEN expression
  [ ELSE expression ]
  ENDIF

sequence-expression :
  sequence-name, [ CURRVAL | NEXTVAL ]
  FROM table-name

operator:
  { + | - | * | / | || | % }

Remarks
Expressions are used in many different places.

Expressions are formed from several different kinds of elements. These are discussed in the sections on functions and variables.

You must be connected to the database to evaluate expressions.

Side effects
None.

See also
  ● “Constants in expressions” on page 22
  ● “Special values” on page 65
  ● “Column names in expressions” on page 23
  ● “SQL functions” on page 143
  ● “Subqueries in expressions” on page 23
  ● “Search conditions” on page 40
  ● “SQL data types” on page 89
  ● “Variables” on page 79
  ● “CASE expressions” on page 24

Standards and compatibility
  ● See the separate descriptions of each class of expression, in the following sections.

Constants in expressions
Constants are numbers or string literals. String constants are enclosed in apostrophes (‘single quotes’). An apostrophe is represented inside a string by two apostrophes in a row.
Column names in expressions

A column name is an identifier preceded by an optional correlation name. A correlation name is usually a table name.

If a column name has characters other than letters, digits and underscore, it must be surrounded by quotation marks (""). For example, the following are valid column names:

- Employees.Name
- address
- "date hired"
- "salary"."date paid"

See also
- “Identifiers” on page 4
- “FROM clause” on page 810

Subqueries in expressions

A subquery is a SELECT statement that is nested inside another SELECT, INSERT, UPDATE, or DELETE statement, or another subquery.

If a subquery matches no rows, it evaluates to NULL.

The SELECT statement must be enclosed in parentheses, and must contain one and only one SELECT list item. When used as an expression, a subquery is generally allowed to return only one value.

A subquery can be used anywhere that a column name can be used. For example, a subquery can be used in the SELECT list of another SELECT statement.

See also
- “Subqueries in search conditions” on page 42

IF expressions

The syntax of the IF expression is as follows:

```
IF condition
THEN expression1
[ ELSE expression2 ]
{ ENIF | END IF }
```

This expression returns the following:

- If `condition` is TRUE, the IF expression returns `expression1`.
- If `condition` is FALSE, the IF expression returns `expression2`.
If condition is FALSE, and there is no expression2, the IF expression returns NULL.

If condition is UNKNOWN, the IF expression returns NULL.

expression1 is evaluated only if condition is TRUE. Similarly, expression2 is evaluated only if condition is FALSE. Both expression1 and expression2 are arbitrary expressions; condition is any valid search condition.

Note
The IF expression is not the same as the IF statement.

Standards and compatibility

- SQL/2008 Vendor extension. The SQL/2008 standard defines the NULLIF, COALESCE, and CASE expressions which can substitute for an IF expression.

See also

- “IF statement” on page 850
- “Search conditions” on page 40
- “NULL special value” on page 71

CASE expressions

The CASE expression provides conditional SQL expressions. Case expressions can be used anywhere an expression can be used.

The syntax of the CASE expression is as follows:

```
CASE expression-1
  WHEN expression-2
  THEN expression-3, ...
  [ ELSE expression-4 ]
  { END | END CASE }
```

If the expression following the CASE clause is equal to the expression following the WHEN clause, then the expression following the THEN statement is returned. Otherwise the expression following the ELSE statement is returned, if it exists.

the CASE expression returns NULL if the ELSE clause doesn't exist and expression-1 doesn't match any of the expression-2...expression-n values.

For example, the following code uses a case expression as the second clause in a SELECT statement.

```
SELECT ID,
  ( CASE Name
      WHEN 'Tee Shirt' then 'Shirt'
      WHEN 'Sweatshirt' then 'Shirt'
      WHEN 'Baseball Cap' then 'Hat'
      ELSE 'Unknown'
      END ) as Type
FROM GROUPO.Products;
```
An alternative syntax is as follows:

```sql
CASE
WHEN search-condition
THEN expression-1, ...
[ ELSE expression-2 ]
END [ CASE ]
```

If the search-condition following the WHEN clause is satisfied, the expression following the THEN statement is returned. Otherwise the expression following the ELSE statement is returned, if it exists.

For example, the following statement uses a case expression as the third clause of a SELECT statement to associate a string with a search-condition.

```sql
SELECT ID, Name,
     ( CASE
         WHEN Name='Tee Shirt' then 'Sale'
         WHEN Quantity >= 50  then 'Big Sale'
         ELSE 'Regular price'
     END ) as Type
FROM GROUPO.Products;
```

**NULLIF function for abbreviated CASE expressions**

The NULLIF function provides a way to write some CASE clauses in short form. The syntax for NULLIF is as follows:

```sql
NULLIF ( expression-1, expression-2 )
```

NULLIF compares the values of the two expressions. If the first expression equals the second expression, NULLIF returns NULL. If the first expression does not equal the second expression, NULLIF returns the first expression.

**Note**

Do not confuse the syntax of the CASE expression with that of the CASE clause.

**Standards and compatibility**

- **SQL/2008**  The CASE expression is a core feature of the SQL/2008 standard. The standard permits any expression referenced by the statement to be evaluated at any point during execution. With SQL Anywhere, expression evaluation occurs when each WHEN clause is evaluated, in their syntactic order, with the exception of constant values that can be determined at compile time.

  Support for END CASE with CASE expressions, in addition to END, is a vendor extension. The SQL/2008 standard defines END for use with CASE expressions and END CASE for use with CASE clauses.

**See also**

- “CASE statement” on page 532
Regular expressions overview

A regular expression is a sequence of characters, wildcards, or operators that defines a pattern to search for within a string. SQL Anywhere supports regular expressions as part of a REGEXP or SIMILAR TO search conditions in the WHERE clause of a SELECT statement, or as an argument to the REGEXP_SUBSTR function. The LIKE search condition does not support regular expressions, although some of the wildcards and operators you can specify with LIKE resemble the regular expression wildcards and operators.

The following SELECT statement uses a regular expression ((K|C[^h])%) to search the Contacts table and return contacts whose last name begins with K or C, but not Ch:

```
SELECT Surname, GivenName
FROM GROUPO.Contacts
WHERE Surname SIMILAR TO '(K|C[^h])%';
```

A regular expression can include additional syntax to specify grouping, quantification, assertions, and alternation, as described below.

- **Grouping** Grouping allows you to group parts of a regular expression to apply some additional matching criteria. For example, '(abc){2}' matches abcabc.

  You can also use grouping to control the order in which the parts of the expression are evaluated. For example, 'ab(cdcd)' looks first for an incidence of cdcd, and then evaluates whether the instance of cdcd is preceded by ab.

- **Quantification** Quantification allows you to control the number of times the preceding part of the expression can occur. For example, a question mark (?) is a quantifier that matches zero or one instance of the previous character. So, 'honou?r' matches both honor and honour.

- **Assertions** Normally, searching for a pattern returns that pattern. Assertions allow you to test for the presence of a pattern, without having that pattern become part of what is returned. For example, 'SQL(?= Anywhere)' matches SQL only if it is followed by a space and then Anywhere.

- **Alternation** Alternation allows you to specify alternative patterns to search for if the preceding pattern cannot be found. Alternate patterns are evaluated from left to right, and searching stops at the first match. For example, 'col(o|ou)r' looks for an instance of color. If no instance is found, colour is searched for instead.

See also

- “Regular expressions syntax” on page 27
- “Search conditions” on page 40
- “LIKE, REGEXP, and SIMILAR TO search conditions” on page 45
- “REGEXP search condition” on page 51
- “SIMILAR TO search condition” on page 53
- “REGEXP_SUBSTR function [String]” on page 329
Regular expressions syntax

Regular expressions are supported with the SIMILAR TO, and REGEXP search conditions, and the REGEXP_SUBSTR function. For SIMILAR TO, regular expression syntax is consistent with the ANSI/ISO SQL standard. For REGEXP and REGEXP_SUBSTR, regular expression syntax and support is consistent with Perl 5.

Regular expressions are used by REGEXP and SIMILAR TO to match a string, whereas regular expressions are used by REGEXP_SUBSTR to match a substring. To achieve substring matching behavior for REGEXP and SIMILAR TO, you can specify wildcards on either side of the pattern you are trying to match. For example, `REGEXP '.*car.*'` matches car, carwash, and vicar. Or, you can rewrite your query to use the REGEXP_SUBSTR function.

Regular expression matching with SIMILAR TO is case- and accent-insensitive. REGEXP and REGEXP_SUBSTR is not affected by the database accent and case sensitivity.

Regular expressions: Metacharacters

Metacharacters are symbols or characters that have a special meaning within a regular expression.

The treatment of metacharacters can vary depending on:

- whether the regular expression is being used with the SIMILAR TO or REGEXP search conditions, or the REGEXP_SUBSTR function
- whether the metacharacter is inside of a character class in the regular expression

Before continuing, you should understand the definition of a character class. A character class is a set of characters enclosed in square brackets, against which characters in a string are matched. For example, in the syntax `SIMILAR TO 'a[b-9]'`, `[1-9]` is a character class and matches one digit in the range of 1 to 9, inclusive. The treatment of metacharacters in a regular expression can vary depending on whether the metacharacter is placed inside a character class. Specifically, most metacharacters are handled as regular characters when positioned inside of a character class.

For SIMILAR TO (only), the metacharacters *, ?, +, _, |, (, ), { must be escaped within a character class.

To include a literal minus sign (-), caret (^), or right-angle bracket (\) character in a character class, it must be escaped.

The list of supported regular expression metacharacters is provided below. Almost all metacharacters are treated the same when used by SIMILAR TO, REGEXP, and REGEXP_SUBSTR:
Left and right square brackets are used to specify a **character class**. A character class is a set of characters to match against.

With the exception of the hyphen (-) and the caret (^), metacharacters and quantifiers (such as * and {m}, respectively) specified within a character class have no special meaning and are evaluated as actual characters.

SQL Anywhere also supports sub-character classes such as POSIX character classes.

<table>
<thead>
<tr>
<th>Character</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ and ]</td>
<td>Left and right square brackets are used to specify a <strong>character class</strong>. A character class is a set of characters to match against. With the exception of the hyphen (-) and the caret (^), metacharacters and quantifiers (such as * and {m}, respectively) specified within a character class have no special meaning and are evaluated as actual characters. SQL Anywhere also supports sub-character classes such as POSIX character classes.</td>
</tr>
<tr>
<td>*</td>
<td>The asterisk can be used to match a character 0 or more times. For example, <code>REGEXP '.*abc'</code> matches a string that ends with abc, and starts with any prefix. So, aabc, xyzabc, and abc match, but bc and abcc do not.</td>
</tr>
<tr>
<td>?</td>
<td>The question mark can be used to match a character 0 or 1 times. For example, <code>'color?our'</code> matches color and colour.</td>
</tr>
<tr>
<td>+</td>
<td>The plus sign can be used to match a character 1 or more times. For example, <code>'bre+'</code> matches bre and bree, but not br.</td>
</tr>
<tr>
<td>-</td>
<td>A hyphen can be used within a character class to denote a range. For example, <code>REGEXP ' [a-e] '</code> matches a, b, c, d, and e.</td>
</tr>
<tr>
<td>%</td>
<td>The percent sign can be used with SIMILAR TO to match any number of characters. The percent sign is not considered a metacharacter for REGEXP and REGEXP_SUBSTR. When specified, it matches a percent sign (%).</td>
</tr>
<tr>
<td>_ (underscore character)</td>
<td>The underscore can be used with SIMILAR TO to match a single character. The underscore is not considered a metacharacter for REGEXP and REGEXP_SUBSTR. When specified, it matches an underscore (_).</td>
</tr>
<tr>
<td></td>
<td>The pipe symbol is used to specify alternative patterns to use for matching the string. In a string of patterns separated by a vertical bar, the vertical bar is interpreted as an <strong>OR</strong> and matching stops at the first match made starting from the leftmost pattern. So, you should list the patterns in descending order of preference. You can specify an unlimited number of alternative patterns.</td>
</tr>
<tr>
<td>( and )</td>
<td>Left and right parenthesis are metacharacters when used for grouping parts of the regular expression. For example, <code>(ab) *</code> matches zero or more repetitions of ab. As with mathematical expressions, you use grouping to control the order in which the parts of a regular expression are evaluated.</td>
</tr>
</tbody>
</table>
Left and right curly braces are metacharacters when used for specifying **quantifiers**. Quantifiers specify the number of times a pattern must repeat to constitute a match. For example:

- **\{m\}** Matches a character exactly \(m\) times. For example, `'519-[0-9]\{3\}-[0-9]\{4\}' matches a phone number in the 519 area code (providing the data is formatted in the manner defined in the syntax).
- **\{m,\}** Matches a character at least \(m\) times. For example, `'[0-9]\{5,\}' matches any string of five or more digits.
- **\{m,n\}** Matches a character at least \(m\) times, but not more than \(n\) times. For example, `SIMILAR TO '_\{5,10\}' matches any string with between 5 and 10 (inclusive) characters.

The backslash is used as an escape character for metacharacters. It can also be used to escape non-metacharacters.

For REGEXP and REGEXP_SUBSTR, when a caret is outside a character class, the caret matches the start of a string. For example, `'^[hc]at'` matches hat and cat, but only at the beginning of the string.

When used inside a character class, the following behavior applies:

- **REGEXP and REGEXP_SUBSTR** When the caret is the first character in a character class, it matches anything other than the characters in the character set. For example, `REGEXP '\[^abc\]'` matches any character other than a, b, or c.

  If the caret is not the first character inside the square brackets, it matches a caret. For example, `REGEXP_SUBSTR '[a-e^c]'` matches a, b, c, d, e, and `^`.

- **SIMILAR TO** For SIMILAR TO, the caret is treated as a subtraction operator. For example, `SIMILAR TO '[a-e^c]'` matches a, b, d, and e.

When used with REGEXP and REGEXP_SUBSTR, matches the end of a string. For example, `REGEXP 'cat$' matches cat, but not catfish.``

When used with REGEXP and REGEXP_SUBSTR, matches any single character. For example, `REGEXP 'a.cd' matches any string of four characters that starts with a and ends with cd.

When used with SIMILAR TO, matches a period (\.)

The colon is used within a character set to specify a subcharacter class. For example, `'[[:alnum:]]'`.
Regular expressions: Special sub-character classes

Sub-character classes are special character classes embedded within a larger character class. In addition to custom character classes where you define the set of characters to match (for example, `[^abxq4]` limits the set of matching characters to a, b, x, q, and 4), SQL Anywhere supports sub-character classes such as most of the POSIX character classes. For example, `[:alpha:]` represents the set of all upper- and lowercase letters.

The REGEXP search condition and the REGEXP_SUBSTR function support all the syntax conventions in the table below, but the SIMILAR TO search expression does not. Conventions supported by SIMILAR TO have a Y in the SIMILAR TO column.

In REGEXP and when using the REGEXP_SUBSTR function, sub-character classes can be negated using a caret. For example, `[^[:alpha:]]` matches the set of all characters except alpha characters.

<table>
<thead>
<tr>
<th>Sub-character class</th>
<th>Additional information</th>
<th>SIMILAR TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>[:alpha:]</td>
<td>Matches upper- and lowercase alphabetic characters in the current collation. For example, <code>'[0-9]{3}[:alpha:]{2}'</code> matches three digits, followed by two letters.</td>
<td>Y</td>
</tr>
<tr>
<td>[:alnum:]</td>
<td>Match digits, and upper- and lowercase alphabetic characters in the current collation. For example, <code>'[:alnum:]+'</code> matches a string of one or more letters and numbers.</td>
<td>Y</td>
</tr>
<tr>
<td>[:digit:]</td>
<td>Match digits in the current collation. For example, <code>'[:digit:]-+'</code> matches a string of one or more digits or dashes. Likewise, <code>'^[:digit:]-+'</code> matches a string of one or more characters that are not digits or dashes.</td>
<td>Y</td>
</tr>
<tr>
<td>[:lower:]</td>
<td>Match lowercase alphabetic characters in the current collation. For example, <code>'[:lower:]'</code> does not match A because A is uppercase.</td>
<td>Y</td>
</tr>
<tr>
<td>[:space:]</td>
<td>Match a single blank (<code>'</code>). For example, the following statement searches Contacts.City for any city with a two word name:</td>
<td></td>
</tr>
</tbody>
</table>
|                     | `SELECT City
|                     | FROM GROUPO.Contacts
|                     | WHERE City REGEXP '.*[:space:]*';` | Y |
| [:upper:]            | Match uppercase alphabetic characters in the current collation. For example, `'[:upper:][ab]'` matches one of: any uppercase letter, a, or b. | Y |
| [:whitespace:]       | Match a whitespace character such as space, tab, formfeed, and carriage return. | Y |
| [:ascii:]            | Match any seven-bit ASCII character (ordinal value between 0 and 127). | Y |
## Sub-character class

<table>
<thead>
<tr>
<th>Sub-character class</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>[:blank:]</td>
<td>Match a blank space, or a horizontal tab. [[:blank:]] is equivalent to [ \t].</td>
</tr>
<tr>
<td>[:cntrl:]</td>
<td>Match ASCII characters with an ordinal value of less than 32, or character value 127 (control characters). Control characters include newline, form feed, backspace, and so on.</td>
</tr>
<tr>
<td>[:graph:]</td>
<td>Match printed characters. [[:graph:]] is equivalent to [[:alnum:][:punct:]].</td>
</tr>
<tr>
<td>[:print:]</td>
<td>Match printed characters and spaces. [[:print:]] is equivalent to [[:graph:][:whitespace:]].</td>
</tr>
<tr>
<td>[:punct:]</td>
<td>Match one of: !&quot;#$%&amp;'()*+,-./:;&lt;=&gt;?@]^_`{</td>
</tr>
<tr>
<td>[:word:]</td>
<td>Match alphabetic, digit, or underscore characters in the current collation. [[:word:]] is equivalent to [[:alnum:]_].</td>
</tr>
<tr>
<td>[:xdigit:]</td>
<td>Match a character that is in the character class [0-9A-Fa-f].</td>
</tr>
</tbody>
</table>

### Regular expressions: Other supported syntax conventions

The following syntax conventions are supported by the REGEXP search condition and the REGEXP_SUBSTR function, and they assume that the backslash is the escape character. These conventions are **not supported by the SIMILAR TO search expression**.

<table>
<thead>
<tr>
<th>Regular expression syntax</th>
<th>Name and meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\0.xxx</td>
<td>Matches the character whose value is \0xxx, where xxx is any sequence of octal digits, and 0 is a zero. For example, \0134 matches a backslash.</td>
</tr>
<tr>
<td>\a</td>
<td>Matches the bell character.</td>
</tr>
<tr>
<td>Regular expression syntax</td>
<td>Name and meaning</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>\A</td>
<td>Used outside a character set to match the start of a string. Equivalent to ^ used outside a character set.</td>
</tr>
<tr>
<td>\b</td>
<td>Matches a backspace character.</td>
</tr>
<tr>
<td>\B</td>
<td>Matches the backslash character ().</td>
</tr>
<tr>
<td>\c X</td>
<td>Matches a named control character. For example, \cZ for ctrl-Z.</td>
</tr>
<tr>
<td>\d</td>
<td>Matches a digit in the current collation. For example, the following statement searches ContactsPhone for all phone numbers that end with 00:</td>
</tr>
<tr>
<td></td>
<td>SELECT Surname, Surname, City, Phone FROM GROUPO.Contacts WHERE Phone REGEXP '\d\d\d\d\d\d\d\d\d\d\d00';</td>
</tr>
<tr>
<td></td>
<td>\d can be used both inside and outside character classes, and is equivalent to [[:digit:]].</td>
</tr>
<tr>
<td>\D</td>
<td>Matches anything that is not a digit. This is the opposite of \d. \D can be used both inside and outside character classes, and is equivalent to [[:digit:]].</td>
</tr>
<tr>
<td></td>
<td>Be careful when using the negated shorthands inside square brackets. [\D\S] is not the same as [^\d\s]. The latter matches any character that is not a digit or whitespace. So it matches x, but not 8. The former, however, matches any character that is either not a digit, or is not whitespace. Because a digit is not whitespace, and whitespace is not a digit, [\D\S] matches any character, digit, whitespace or otherwise.</td>
</tr>
<tr>
<td>\e</td>
<td>Matches the escape character.</td>
</tr>
<tr>
<td>\E</td>
<td>Ends the treatment of metacharacters as non-metacharacters, initiated by a \Q.</td>
</tr>
<tr>
<td>\f</td>
<td>Matches a form feed.</td>
</tr>
<tr>
<td>Regular expression syntax</td>
<td>Name and meaning</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>\n</td>
<td>Matches a new line.</td>
</tr>
<tr>
<td>\Q</td>
<td>Treat all metacharacters as non-metacharacters, until \E is encountered. For example, \Q 5 \E is equivalent to [ $.</td>
</tr>
<tr>
<td>\r</td>
<td>Matches a carriage return.</td>
</tr>
<tr>
<td>\s</td>
<td>Matches a space or a character treated as whitespace. For example, the following statement returns all product names from Products.Product-Name that have at least one space in the name: SELECT Name FROM GROUPO.Products WHERE Name REGEXP '.<em>\s.</em>' \s can be used both inside and outside character classes, and is equivalent to [[:white-space:]].</td>
</tr>
<tr>
<td>\S</td>
<td>Matches a non-whitespace character. This is the opposite of \s, and is equivalent to [^[:white-space:]]. \S can be used both inside and outside character classes. Be careful when using the negated shorthands inside square brackets. [\D\S] is not the same as [^\d\s]. The latter matches any character that is not a digit or whitespace. So it matches x, but not 8. The former, however, matches any character that is either not a digit, or is not whitespace. Because a digit is not whitespace, and whitespace is not a digit, [\D\S] matches any character, digit, whitespace or otherwise.</td>
</tr>
<tr>
<td>\t</td>
<td>Matches a horizontal tab.</td>
</tr>
<tr>
<td>\v</td>
<td>Matches a vertical tab.</td>
</tr>
</tbody>
</table>
### Regular expression syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Name and meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\w</code></td>
<td>Matches an alphabetic character, digit, or underscore in the current collation. For example, the following statement returns all surnames from Contacts.Surname that are exactly seven alphanumeric characters in length: SELECT Surname FROM GROUPO.Contacts WHERE Surname REGEXP '\w{7}'; <code>\w</code> can be used both inside and outside character classes. Equivalent to <code>[:alnum:][_]</code>.</td>
</tr>
<tr>
<td><code>\W</code></td>
<td>Matches anything that is not an alphabetic character, digit, or underscore in the current collation. This is the opposite of <code>\w</code>, and is equivalent to <code>[^[:alnum:][_]]</code>. This regular expression can be used both inside and outside character classes.</td>
</tr>
<tr>
<td><code>\x hh</code></td>
<td>Matches the character whose value is 0xhh, where hh is, at most, two hex digits. For example, <code>\x2D</code> is equivalent to a hyphen. Equivalent to <code>\{hh}</code>.</td>
</tr>
<tr>
<td><code>\x{ hhh }</code></td>
<td>Matches the character whose value is 0xhhh, where hhh is, at most, eight hex digits.</td>
</tr>
<tr>
<td><code>\z</code> and <code>\Z</code></td>
<td>Matches the position (not character) at the end of the string. Equivalent to <code>$</code>.</td>
</tr>
</tbody>
</table>

### Regular expressions: Assertions

Assertions test whether a condition is true, and affect the position in the string where matching begins. Assertions do not return characters; the assertion pattern is not included in the final match. These assertions are supported by the REGEXP search condition and the REGEXP_SUBSTR function. These conventions are not supported by the SIMILAR TO search expression.

Lookahead and lookbehind assertions can be useful with REGEXP_SUBSTR when trying to split a string. For example, you can return the list of street names (without the street numbers) in the Address column of the Customers table by executing the following statement:
SELECT REGEXP_SUBSTR( Street, '(?<="\S+\s+).*' )
FROM GROUPO.Customers;

Another example is using a regular expression to verify that a password conforms to certain rules. You could use a zero width assertion similar to the following:

```
IF password REGEXP '(?=.*[[:digit:]])(?=.*[[:alpha:]].*[[:alpha:]])[^[:word:]] {4,12}' THEN
   MESSAGE 'Password conforms' TO CLIENT;
ELSE
   MESSAGE 'Password does not conform' TO CLIENT;
END IF
```

The password is valid when the following are true:

- **password** has at least one digit (zero width positive assertion with [[:digit:]])
- **password** has at least two alphabetic characters (zero width positive assertion with [[:alpha:]].*[[:alpha:]])
- **password** contains only alpha-numeric or underscore characters ([[:word:]])
- **password** is at least 4 characters, and at most 12 characters ({4,12})

The following table contains the assertions supported by SQL Anywhere:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(?= pattern )</td>
<td>Positive lookahead zero-width assertion</td>
</tr>
<tr>
<td></td>
<td>Looks to see if the current position in the</td>
</tr>
<tr>
<td></td>
<td>string is immediately followed by an occurrence</td>
</tr>
<tr>
<td></td>
<td>of pattern, without pattern becoming part of</td>
</tr>
<tr>
<td></td>
<td>the match string. 'A (?=B)' matches an A that</td>
</tr>
<tr>
<td></td>
<td>is followed by a B, without making the B part</td>
</tr>
<tr>
<td></td>
<td>of the match.</td>
</tr>
<tr>
<td></td>
<td>For example, SELECT REGEXP_SUBSTR( 'in new york city', 'new(?=.*\ syork)' ); returns the substring new since it is immediately followed by 'york' (note the space before york).</td>
</tr>
<tr>
<td>(?! pattern )</td>
<td>Negative lookahead zero-width assertions</td>
</tr>
<tr>
<td></td>
<td>Looks to see if the current position in the</td>
</tr>
<tr>
<td></td>
<td>string is not immediately followed by an</td>
</tr>
<tr>
<td></td>
<td>occurrence of pattern, without pattern</td>
</tr>
<tr>
<td></td>
<td>becoming part of the match string. So, 'A (!B)'</td>
</tr>
<tr>
<td></td>
<td>matches an A that is not followed by a B.</td>
</tr>
<tr>
<td></td>
<td>For example, SELECT REGEXP_SUBSTR('new jersey', 'new(?!\ syork)' ); returns the substring new.</td>
</tr>
<tr>
<td>(?&lt;= pattern )</td>
<td>Positive lookbehind zero-width assertions</td>
</tr>
<tr>
<td></td>
<td>Looks to see if the current position in the</td>
</tr>
<tr>
<td></td>
<td>string is immediately preceded by an occurrence</td>
</tr>
<tr>
<td></td>
<td>of pattern, without pattern becoming part of</td>
</tr>
<tr>
<td></td>
<td>the match string. So, '(?&lt;=&quot;\S+\s+\s+\s+york') matches a B that is immediately preceded by an A, without making A part of the match.</td>
</tr>
<tr>
<td></td>
<td>For example, SELECT REGEXP_SUBSTR('new york', '(?&lt;=&quot;\S+\s+\s+\s+york') ); returns the substring york.</td>
</tr>
</tbody>
</table>
### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| (?<! pattern) | **Negative lookbehind zero-width assertions**<br>Literally checks if the current position in the string is *not* immediately preceded by an occurrence of `pattern`, without `pattern` becoming part of the match string.  
For example, `SELECT REGEXP_SUBSTR('about york', '(?<!new\s)york');` returns the substring york. |
| (?> pattern)  | **Possessive local subexpression**<br>Matches only the largest prefix of the remaining string that matches `pattern`.  
For example, in `'aa' REGEXP '(?>a*)a'`, `(?>a*)` matches (and consumes) the aa, and never just the leading a. As a result, `'aa' REGEXP '(?>a*)a'` evaluates to false. |
| (?: pattern)  | **Non-capturing block**<br>This is functionally equivalent to just `pattern`, and is provided for compatibility.  
For example, in `'bb' REGEXP '(?::b*)b'`, `(?::b*)` matches (and consumes) the bb. However, unlike possessive local subexpression, the last b in bb is given up to allow the whole match to succeed (that is, to allow the matching to the b found outside the non-capturing block).  
Likewise, `'a (?::bc|b) c'` matches abcc, and abc. In matching abc, backtracking on the final c in bc takes place so that the c outside the group can be used to make the match successful. |
| (?# text)   | Used for comments. The content of `text` is ignored. |

### See also

- “Regular expressions examples” on page 36
- “REGEXP_SUBSTR function [String]” on page 329
- “LIKE, REGEXP, and SIMILAR TO: Differences in character comparisons” on page 46
- “Regular expressions: Metacharacters” on page 27
- “Regular expressions: Special sub-character classes” on page 30

### Regular expressions examples

The following table shows example uses of regular expressions. All examples work for REGEXP and some also work for SIMILAR TO, as noted in the Example column. Results vary depending on the search condition you use for searching. For those that work with SIMILAR TO, results can vary further depending on case and accent sensitivity.

Backslashes should be doubled if the examples are used in literal strings (for example, `\.+@+.\+\..+`)
### Credit Card Numbers (REGEXP only):

**Visa:**
\[4[0-9]{3}\ \s[0-9]{4}\ \s[0-9]{4}\ \s[0-9]{4}\]

**MasterCard:**
\[5[0-9]{3}\ \s[0-9]{4}\ \s[0-9]{4}\ \s[0-9]{4}\]

**American Express:**
\[37[0-9]{2}\ \s[0-9]{4}\ \s[0-9]{4}\ \s[0-9]{4}\]

**Discover:**
\[6011\ \s[0-9]{4}\ \s[0-9]{4}\ \s[0-9]{4}\]

### Dates (REGEXP and SIMILAR TO):

\([0-2]\[0-9]\ |30|31)/(0[1-9]|1[0-2])/[0-9]{4}\]

### Windows absolute paths (REGEXP only):

\([A-Za-z]:\|:\|[:alnum:]\|[:whitespace:]!\#\$&'()+,-.\;=@\|\^\_`{}~.]*\]

### Email Addresses (REGEXP only):

\[[:word:]-.]+@[:word:]-.+.+

### HTML Hexadecimal Color Codes (REGEXP only):

\[A-F0-9\] \{6\}

### Sample matches

**Visa:** Matches (Visa): 4123 6453 2222 1746
Non-Matches (Visa):
3124 5675 4400 4567, 4123-6453-2222-1746

**MasterCard:** Similarly, MasterCard matches a set of 16 numbers, starting with 5, with a space between each subset of four numbers. American Express and Discover are the same, but must start with 37 and 6011 respectively.

**American Express:**

**Discover:**

**Dates:** Matches: 31/04/1999, 15/12/4567

**Windows absolute paths:** Matches: \server\share\file
Non-Matches: \directory\directory2, /directory2

**Email Addresses:** Matches: abc.123@def456.com, _123@abc.ca
Non-Matches: abc@dummy, ab*cd@efg.hijkl

**HTML Hexadecimal Color Codes:** Matches: AB1234, CCCCCC, 12AF3B
Non-Matches: 123G45, 12-44-CC
<table>
<thead>
<tr>
<th>Example</th>
<th>Sample matches</th>
</tr>
</thead>
</table>
| **IP Addresses (REGEXP only):**  
\((2(5[0-5]|([0-4][0-9])|1([0-9][0-9]| \((1-9)[0-9]|([0-9])\))\.(2(5[0-5]|([0-4][0-9])|1([0-9][0-9]|([1-9][0-9])|([0-9])\))\.(2(5[0-5]|([0-4][0-9])|1([0-9][0-9]|([1-9][0-9])|([0-9])\))\))  
| Matches: 10.25.101.216  
Non-Matches: 0.0.0, 256.89.457.02 |
| **Java Comments (REGEXP only):**  
\(/\*.*\*/|//[^
\]/*  
| Matches Java comments that are between /* and */ or one line comments prefaced by //  
Non-Matches: a=1 |
| **Money (REGEXP only):**  
\((-|\+)?\$[0-9]*\.[0-9]{2}  
| Matches: $1.00, -$97.65  
Non-Matches: $1, 1.00$, $-75.17 |
| **Positive, negative numbers, and decimal values (REGEXP only):**  
\((-|\+)?[0-9]+(\.[0-9]+)?  
| Matches: +41, -412, 2, 7968412, 41, +41.1, -3.141592653  
Non-Matches: ++41, 41.1.19, +97.14 |
| **Passwords (REGEXP and SIMILAR TO):**  
\[[:alnum:]\]{4,10}  
| Matches: abcd, 1234, A1b2C3d4, 1a2B3  
Non-Matches: abc, *ab12, abcdefghijkl |
| **Passwords (REGEXP only):**  
\[a-zA-Z\]\w\{3,7\}  
| Matches: AB_cd, A1_b2c3, a123_  
Non-Matches: *&^g, abc, 1bcd |
| **Phone Numbers (REGEXP and SIMILAR TO):**  
\((2-9)[0-9]\((2-9)\[0-9]\((2-9)\[0-9]\((2-9)(2)\[0-9]\((2)\[0-9]\((2)\[0-9]\((4)\))\)\))\((2-9)\[0-9]\((2)\[0-9]\((2)\[0-9]\((4)\))\)\)  
| Matches: 519-883-6898, 519 888 6898  
Non-Matches: 888 6898, 5198886898, 519 883-6898 |
| **Sentences (REGEXP only):**  
\[A-Z0-9\].*(\.|\?|!)  
| Matches: Hello, how are you?  
Non-Matches: i am fine |
| **Sentences (REGEXP only):**  
\[:upper:]0-9\.*[\.?!]  
| Matches: Hello, how are you?  
Non-Matches: i am fine |
| **Social Security Numbers (REGEXP and SIMILAR TO):**  
\[0-9\](3)-[0-9]\((2)-[0-9]\((4)\)  
| Matches: 123-45-6789  
Non-Matches:123 45 6789, 123456789, 1234-56-7891 |
Example

<table>
<thead>
<tr>
<th>URLs (REGEXP only):</th>
<th>Sample matches</th>
</tr>
</thead>
</table>

See also

- “Regular expressions syntax” on page 27
- “LIKE, REGEXP, and SIMILAR TO search conditions” on page 45

Compatibility of expressions

SQL Anywhere employs the SQL/2008 convention, that strings enclosed in apostrophes are constant expressions, and strings enclosed in quotation marks (double quotes) are delimited identifiers (names for database objects).

The quoted_identifier option

SQL Anywhere provides a quoted_identifier option that allows the interpretation of delimited strings to be changed. By default, the quoted_identifier option is set to On in SQL Anywhere.

You cannot use SQL reserved words as identifiers if the quoted_identifier option is Off.

Setting the option

The following statement changes the setting of the quoted_identifier option to On:

```
SET quoted_identifier On;
```

The following statement changes the setting of the quoted_identifier option to Off:

```
SET quoted_identifier Off;
```

Compatible interpretation of delimited strings

You can choose to use either the SQL/2008 or the default Transact-SQL convention in SQL Anywhere as long as the quoted_identifier option is set to the same value in each DBMS.

Examples

If you choose to operate with the quoted_identifier option On (the default setting), then the following statements involving the SQL keyword `user` are valid for both DBMSs.

```
CREATE TABLE "user" ( col1 char(5) )
go
INSERT "user" ( col1 )
VALUES ( 'abcde' )
go
```
If you choose to operate with the quoted_identifier option off then the following statement is valid for both DBMSs. In the following example, Chin is a string and not an identifier.

```sql
SELECT *
FROM GROUPO.Employees
WHERE Surname = "Chin"
go
```

See also
- “quoted_identifier option” [SQL Anywhere Server - Database Administration]
- “Reserved words” on page 1

**Search conditions**

A search condition is the criteria specified for a WHERE clause, a HAVING clause, a CHECK clause, an ON phrase in a join, or an IF expression. A search condition is also called a predicate.

**Syntax**

```
search-condition : expression comparison-operator expression
 expression comparison-operator [ [ ANY | SOME | ALL ] ( subquery )
 expression IS [ NOT ] DISTINCT FROM expression
 expression IS [ NOT ] NULL
 expression [ NOT ] BETWEEN expression AND expression
 expression [ NOT ] LIKE pattern [ ESCAPE expression ]
 expression [ NOT ] SIMILAR TO pattern [ ESCAPE escape-expression ]
 expression [ NOT ] REGEXP pattern [ ESCAPE escape-expression ]
 expression [ NOT ] IN ( expression , ... )
 [ ( query-expression )
 NOT search-condition
 CONTAINS (column-name [,... ] , query-string )
 EXISTS ( query-expression )
 search-condition [ { AND | OR } search-condition ] [ ... ]
 ( search-condition , estimate )
 search-condition IS [ NOT ] { TRUE | FALSE | UNKNOWN }
 expression IS [ NOT ] OF ( { ONLY } type-name ,... )
 trigger-operation
```

**comparison-operator :**

```
= | >= | <= | < > | != | < | > | !=
```

**trigger-operation :**

```
INSERTING
| DELETING
```
| UPDATING [ ( column-name-string ) ] |
| UPDATE( column-name ) |

Parameters

- ALL search condition
- ANY and SOME search conditions
- IS [NOT] DISTINCT FROM search condition
- BETWEEN search condition
- CONTAINS search condition
- EXISTS search condition
- LIKE search condition
- SIMILAR TO search condition
- REGEXP search condition
- IS OF type-expression, and IS NOT OF type-expression
  This type predicate was added for support of spatial geometries, but it can be used for any existing data type as well.

Remarks

Search conditions are used either to choose a subset of the rows from a table, or in a control statement such as an IF statement to determine control of flow.

In SQL, every condition evaluates as one of TRUE, FALSE, or UNKNOWN. This is called three-valued logic. The result of a comparison is UNKNOWN if either value being compared is the NULL value.

Rows satisfy a search condition only if the result of the condition is TRUE. Rows for which the condition is UNKNOWN or FALSE do not satisfy the search condition.

Subqueries form an important class of expression that is used in many search conditions.

The LIKE, SIMILAR TO, and REGEXP search conditions are very similar.

Prerequisites

- Must be connected to the database.

Side effects

- None.
Subqueries in search conditions

Subqueries that return exactly one column and either zero or one row can be used in any SQL statement wherever a column name could be used, including in the middle of an expression.

For example, expressions can be compared to subqueries in comparison conditions as long as the subquery does not return more than one row. If the subquery (which must have exactly one column) returns one row, then the value of that row is compared to the expression. If a subquery returns no rows, the value of the subquery is NULL.

Subqueries that return exactly one column and any number of rows can be used in IN, ANY, ALL, and SOME search conditions. Subqueries that return any number of columns and rows can be used in EXISTS search conditions. These search conditions are discussed in the following sections.

Standards and compatibility

- **SQL/2008** The use of a scalar subquery as an arbitrary expression is a core feature of the SQL/2008 standard.

See also

- “Comparison operators” on page 9

**ALL search condition**

Syntax

```
expression comparison-operator ALL ( subquery )
```

comparison-operator:

```
= 
| AND |
| OR  |
```

See also

- “Expressions” on page 21
- “ALL search condition” on page 42
- “ANY and SOME search conditions” on page 43
- “BETWEEN search condition” on page 45
- “CONTAINS search condition” on page 55
- “EXISTS search condition” on page 62
- “LIKE search condition” on page 47
- “SIMILAR TO search condition” on page 53
- “REGEXP search condition” on page 51
- “Spatial data type syntax” [SQL Anywhere Server - Spatial Data Support]
- “Three-valued logic” on page 63
- “NULL special value” on page 71
- “Subqueries in search conditions” on page 42
- “LIKE, REGEXP, and SIMILAR TO search conditions” on page 45
Remarks

With the ALL search condition, if the value of subquery result set is the empty set, the search condition evaluates to TRUE. Otherwise, the search condition evaluates to TRUE, FALSE, or UNKNOWN, depending on the value of `expression`, and the result set returned by the subquery, as follows:

<table>
<thead>
<tr>
<th>If the expression value is...</th>
<th>and the result set returned by the subquery contains at least one NULL, then...</th>
<th>or the result set returned by the subquery contains no NULLs, then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>not NULL</td>
<td>If there exists at least one value in the subquery result set for which the comparison with the expression value is FALSE, then the search condition evaluates to FALSE. Otherwise, the search condition evaluates to UNKNOWN.</td>
<td>If there exists at least one value in the subquery result set for which the comparison with the expression value is FALSE, then the search condition evaluates to FALSE. Otherwise, the search condition evaluates to TRUE.</td>
</tr>
</tbody>
</table>

Standards and compatibility

- SQL/2008 Core feature.

**ANY and SOME search conditions**

**Syntax**

```
expression comparison-operator { ANY | SOME } ( subquery )
```

**Remarks**

The keywords ANY and SOME are synonymous.

With the ANY and SOME search conditions, if the subquery result set is the empty set, the search condition evaluates to FALSE. Otherwise, the search condition evaluates to TRUE, FALSE, or
UNKNOWN, depending on the value of `expression`, and the result set returned by the subquery, as follows:

<table>
<thead>
<tr>
<th>If the expression value is...</th>
<th>and the result set returned by the subquery contains at least one NULL, then...</th>
<th>or the result set returned by the subquery contains no NULLs, then...</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>not NULL</td>
<td>If there exists at least one value in the subquery result set for which the comparison with the expression value is TRUE, then the search condition evaluates to TRUE. Otherwise, the search condition evaluates to UNKNOWN.</td>
<td>If there exists at least one value in the subquery result set for which the comparison with the expression value is TRUE, then the search condition evaluates to TRUE. Otherwise, the search condition evaluates to FALSE.</td>
</tr>
</tbody>
</table>

An ANY or SOME search condition with an equality operator, evaluates to TRUE if `expression` is equal to any of the values in the result of the subquery, and FALSE if the value of the expression is not NULL, does not equal any of the values in the result of the subquery, and the result set doesn't contain NULLs.

**Note**
The usage of `= ANY` or `= SOME` is equivalent to using the IN keyword.

### Standards and compatibility

- SQL/2008 Core feature.

### IS DISTINCT FROM and IS NOT DISTINCT FROM search conditions

**Syntax**

```
expression1 IS [ NOT ] DISTINCT FROM expression2
```

**Remarks**

The IS DISTINCT FROM and IS NOT DISTINCT FROM search conditions are sargable and evaluate to TRUE or FALSE.

The IS NOT DISTINCT FROM search condition evaluates to TRUE if `expression1` is equal to `expression2`, or if both expressions are NULL. This is equivalent to a combination of two search conditions, as follows:

```
expression1 = expression2 OR ( expression1 IS NULL AND expression2 IS NULL )
```

The IS DISTINCT FROM syntax reverses the meaning. That is, IS DISTINCT FROM evaluates to TRUE if `expression1` is not equal to `expression2`, and at least one of the expressions is not NULL. This is equivalent to the following:
\[
\text{NOT ( expression1 = expression2 OR ( expression1 IS NULL AND expression2 IS NULL ))}
\]

Standards and compatibility

- **SQL/2008**  The IS [NOT] DISTINCT FROM predicate is defined in SQL/2008 standard. The IS DISTINCT FROM predicate is Feature T151, "DISTINCT predicate", of the SQL/2008 standard. The IS NOT DISTINCT FROM predicate is Feature T152, "DISTINCT predicate with negation", of the SQL/2008 standard.

See also

- “Query predicates” [*SQL Anywhere Server - SQL Usage*]

**BETWEEN search condition**

**Syntax**

\[expression\ [\text{ NOT }] \text{ BETWEEN } \text{start-expression AND end-expression}\]

**Remarks**

The BETWEEN search condition can evaluate as TRUE, FALSE, or UNKNOWN. Without the NOT keyword, the search condition evaluates as TRUE if \(expression\) is between \text{start-expression} and \text{end-expression}. The NOT keyword reverses the meaning of the search condition but leaves UNKNOWN unchanged.

The BETWEEN search condition is equivalent to a combination of two inequalities:

\[\text{[ NOT ]} (\ expression \geq \text{start-expression AND expression} \leq \text{end-expression} )\]

Standards and compatibility

- **SQL/2008**  Core feature.

**LIKE, REGEXP, and SIMILAR TO search conditions**

The REGEXP, LIKE, and SIMILAR TO search conditions are similar in that they all attempt to match a pattern to a string. Also, all three attempt to match an entire string, not a substring within the string.

The basic syntax for all three search conditions is similar:

\[expression \text{ search-condition pattern}\]

**LIKE, REGEXP, and SIMILAR TO: Differences in pattern definition**

- REGEXP supports a superset of regular expression syntax supported by SIMILAR TO. In addition, for compatibility with other products, the REGEXP search condition supports several syntax extensions. Also, REGEXP and SIMILAR TO have a different default escape character and process the characters underscore (\_\_), percent (\%\_), and caret (\^\_) differently. REGEXP behavior matches closely with Perl 5 (except where Perl syntax and operators are not supported).
LIKE syntax for pattern is simple and supports a small set of wildcards, but does not support the full regular expression syntax.

SIMILAR TO syntax for pattern allows a robust pattern matching using the regular expression syntax defined in the ANSI/ISO SQL standard.

LIKE, REGEXP, and SIMILAR TO: Differences in character comparisons

When performing comparisons, REGEXP behavior is different from LIKE and SIMILAR TO. For REGEXP comparisons, the database server uses code point values in the database character set for comparisons. This is consistent with other regular expression implementations such as Perl.

For LIKE and SIMILAR TO, the database server uses the equivalence and sort order in the database collation for comparisons. This is consistent with how the database evaluates comparison operators such as > and =.

The difference in character comparison methods means that results for matching and range evaluation for REGEXP and LIKE/SIMILAR differ as well.

Differences in matching

Since REGEXP uses code point values, it only matches a literal in a pattern if it is the exact same character. REGEXP matching is therefore not impacted by such things as database collation, case-sensitivity, or accent sensitivity. For example, 'A' could never be returned as a match for 'a'.

Since LIKE and SIMILAR TO use the database collation, results are impacted by case- and accent-sensitivity when determining character equivalence. For example, if the database collation is case- and accent-insensitive, matches are case- and accent-insensitive. So, an 'A' could be returned as a match for 'a'.

Differences in range evaluation

Since REGEXP uses code points for range evaluation, a character is considered to be in the range if its code point value is equal to, or between, the code point values for the start and end of the range. For example, the comparison \texttt{x \ REGEXP '[A-C]'}, for the single character \texttt{x}, is equivalent to \texttt{CAST(x AS BINARY) >= CAST(A AS BINARY) AND CAST(x AS BINARY) <= CAST(C AS BINARY)}.

Since LIKE and SIMILAR TO use the collation sort order for range evaluation, a character is considered to be in the range if its position in the collation is the same as, or between, the position of the start and end characters for the range. For example, the comparison \texttt{x \ SIMILAR TO '[A-C]' (where x is a single character)} is equivalent to \texttt{x >= A AND x <= C}, and the comparison operators are evaluated using the collation sort ordering.

The following table shows the set of characters included in the range \texttt{'[A-C]'} as evaluated by LIKE, SIMILAR TO, and REGEXP. Both databases use the 1252LATIN1 collation, but the first database is case-insensitive, while the second one is case sensitive.

<table>
<thead>
<tr>
<th>demo.db (case-insensitive)</th>
<th>LIKE/SIMILAR TO '[A-C]'</th>
<th>REGEXP '[A-C]'</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,B,C,a,b,c,*À,Á,Ä,Â,Å,Æ,Ç,è,é,è,ë,â,ã,œ,ç</td>
<td>A, B, C</td>
<td></td>
</tr>
</tbody>
</table>
The following can be observed in the results:

- LIKE and SIMILAR TO include accented characters in the range.
- LIKE and SIMILAR TO include different characters depending on database case-sensitivity. Specifically, they include any lowercase letters found within the range, which you may not have anticipated when searching on a case-sensitive database.
  
  Similarly, on a case-sensitive database, some characters included in the range might appear to be inconsistent. For example, SIMILAR TO '[A-C]' on a case-sensitive database includes A, b, B, c, C but not a because a occurs before the uppercase A in the sort order.
- REGEXP returns only A, B, C regardless of database case sensitivity. If you want the range to include lowercase letters, you must add them to the range definition. For example, REGEXP '[a-zA-C]' .
- the REGEXP set of characters does not change, regardless of database case-sensitivity.

Even though your database uses a different collation, or has different case- or accent-sensitivity settings than the examples above, you can perform a similar test to see what is returned by LIKE, SIMILAR TO, or REGEXP by connecting to the database and executing any of these statements:

```
SELECT CHAR( row_num ) FROM RowGenerator WHERE CHAR( row_num ) LIKE '[A-C]';
SELECT CHAR( row_num ) FROM RowGenerator WHERE CHAR( row_num ) REGEXP '[A-C]';
SELECT CHAR( row_num ) FROM RowGenerator WHERE CHAR( row_num ) SIMILAR TO '[A-C]';
```

See also
- “Regular expressions overview” on page 26
- “Regular expressions syntax” on page 27
- “Regular expressions examples” on page 36

### LIKE search condition

#### Syntax

The syntax for the LIKE search condition is as follows:

```
expression [ NOT ] LIKE pattern [ ESCAPE escape-character ]
```

#### Parameters

- **expression** The string to be searched.
• **pattern**   The pattern to search for within *expression*.

• **escape-character**   The character to use to escape special characters such as underscores and percent signs.

**Remarks**

The LIKE search condition attempts to match *expression* with *pattern* and evaluates to TRUE, FALSE, or UNKNOWN.

The search condition evaluates to TRUE if *expression* matches *pattern* (assuming NOT was not specified). If either *expression* or *pattern* is the NULL value, the search condition evaluates to UNKNOWN. The NOT keyword reverses the meaning of the search condition, but leaves UNKNOWN unchanged.

*expression* is interpreted as a CHAR or NCHAR string. The entire contents of *expression* is used for matching. Similarly, *pattern* is interpreted as a CHAR or NCHAR string and can contain any number of the supported wildcards from the following table:

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ (underscore)</td>
<td>Any one character. For example, a_ matches ab and ac, but not a.</td>
</tr>
<tr>
<td>% (percent)</td>
<td>Any string of zero or more characters. For example, bl% matches bl and bla.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Any single character in the specified range or set. For example, T[o][i]m matches Tom or Tim.</td>
</tr>
<tr>
<td>[^]</td>
<td>Any single character <em>not</em> in the specified range or set. For example, M[^c] matches Mb and Md, but not Mc.</td>
</tr>
</tbody>
</table>

All other characters must match exactly.

For example, the following search condition returns TRUE for any row where name starts with the letter a and has the letter b as its second last character.

```
... name LIKE 'a%b_'
```

If *escape-character* is specified, it must evaluate to a single-byte CHAR or NCHAR character. The escape character can precede a percent, an underscore, a left square bracket, or another escape character in the *pattern* to prevent the special character from having its special meaning. When escaped in this manner, a percent matches a percent, and an underscore matches an underscore.

All patterns of 126 bytes or less are supported. Patterns of greater than 126 bytes that do not contain wildcards are not supported. The number of bytes used to represent the pattern depends on whether the pattern is CHAR or NCHAR.
**Different ways to use the LIKE search condition**

<table>
<thead>
<tr>
<th>To search for</th>
<th>Example</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>One of a set of characters</td>
<td>LIKE 'sm[iy]th'</td>
<td>A set of characters to look for is specified by listing the characters inside square brackets. In this example, the search condition matches <em>smith</em> and <em>smyth</em>.</td>
</tr>
</tbody>
</table>
| One of a range of characters | LIKE '^[a-r]ough' | A range of characters to look for is specified by giving the ends of the range inside square brackets, separated by a hyphen. In this example, the search condition matches *bough* and *rough*, but not *tough*.  

The range of characters [a-z] is interpreted as "greater than or equal to a, and less than or equal to z", where the greater than and less than operations are carried out within the collation of the database.  

The lower end of the range must precede the higher end of the range. For example, [z-a] does not match anything because no character matches the [z-a] range. |
| Ranges and sets combined | ... LIKE '^[a-r]ough' | You can combine ranges and sets within square brackets. In this example, ... LIKE '^[a-rt]ough' matches *bough*, *rough*, and *tough*.  

The pattern [a-rt] is interpreted as exactly one character that is either in the range a to r inclusive, or is t. |
| One character not in a range | ... LIKE '^[^a-r]ough' | The caret character (^) is used to specify a range of characters that is excluded from a search. In this example, LIKE '^[^a-rt]ough' matches the string *tough*, but not the strings *rough* or *bough*.  

The caret negates the rest of the contents of the brackets. For example, the bracket [^a-rt] is interpreted as exactly one character that is not in the range a to r inclusive, and is not t. |
| Search patterns with trailing blanks | '90 ', '90[ ]' and '90_' | When your search pattern includes trailing blanks, the database server matches the pattern only to values that contain blanks—it does not blank pad strings. For example, the patterns '90 ', '90[ ]', and '90_' match the expression '90', but do not match the expression '90', even if the value being tested is in a CHAR or VARCHAR column that is three or more characters in width. |

**Special cases of ranges and sets**

Any single character in square brackets means that character. For example, [a] matches just the character a. [^] matches just the caret character, [%] matches just the percent character (the percent
character does not act as a wildcard in this context), and [ ] matches just the underscore character. Also, 
[ [ ]} matches just the character [.

Other special cases are as follows:

- The pattern [ a-] matches either of the characters a or -.
- The pattern [ ] is never matched and always returns no rows.
- The patterns [ or [abp-q return syntax errors because they are missing the closing bracket.
- You cannot use wildcards inside square brackets. The pattern [a%b] finds one of a, %, or b.
- You cannot use the caret character to negate ranges except as the first character in the bracket. The pattern [a ^b] finds one of a, ^, or b.

Case sensitivity and how comparisons are performed

If the database collation is case sensitive, the search condition is also case sensitive. To perform a case insensitive search with a case sensitive collation, you must include upper and lower characters. For example, the following search condition evaluates to true for the strings Bough, rough, and TOUGH:

```sql
LIKE '[a-zA-Z][oO][uU][gG][hH]'  
```

Comparisons are performed character by character, unlike the equivalence (=) operator and other operators where the comparison is done string by string. For example, when a comparison is done in a UCA collation (CHAR or NCHAR with the collation set to UCA), 'Æ' = 'Æ' is true, but 'Æ' LIKE 'Æ' is false.

For a character-by-character comparison to match, each single character in the expression being searched must match a single character (using the collation's character equivalence), or a wildcard in the LIKE expression.

National character (NCHAR) support

LIKE search conditions can be used to compare CHAR and NCHAR strings. In this case, character set conversion is performed so that the comparison is done using a common data type. Then, a character-by-character comparison is performed.

You can specify expression or pattern as an NCHAR string literal by prefixing the quoted value with N (for example, expression LIKE N'pattern'). You can also use the CAST function to cast the pattern to CHAR or NCHAR (for example, expression LIKE CAST(pattern AS datatype).

Blank padded databases

The semantics of a LIKE pattern does not change if the database is blank-padded since matching expression to pattern involves a character-by-character comparison in a left-to-right fashion. No additional blank padding is performed on the value of either expression or pattern during the evaluation. Therefore, the expression a1 matches the pattern a1, but not the patterns 'a1 ' (a1, with a space after it) or a1_.


Standards and compatibility

- **SQL/2008** The LIKE search condition is a core feature of the SQL/2008 standard. However, there are subtle differences in behavior from that of the standard because SQL Anywhere supports case-insensitive collations and blank-padding.

  SQL Anywhere supports optional SQL language feature F281, which permits the pattern and escape-expressions to be arbitrary expressions evaluated at execution time. Feature F281 also permits expression to be an expression more complex than a simple column reference.

  The use of character ranges and sets contained in square brackets [ ] is a vendor extension.

  SQL Anywhere supports SQL/2008 feature T042, which permits LIKE search conditions to reference string-expressions that are LONG VARCHAR values.

  LIKE search conditions that specify NCHAR string expressions or patterns is optional SQL language feature F421 of the ANSI SQL/2008 standard.

See also

- “Comparisons between CHAR and NCHAR” on page 132
- “String literals” on page 8
- “CAST function [Data type conversion]” on page 174
- “LIKE, REGEXP, and SIMILAR TO: Differences in character comparisons” on page 46
- “LIKE, REGEXP, and SIMILAR TO search conditions” on page 45
- “The WHERE clause: Specifying rows” [SQL Anywhere Server - SQL Usage]
- “REGEXP search condition” on page 51
- “SIMILAR TO search condition” on page 53

REGEXP search condition

Matches a pattern against a string.

Syntax

```
expression [ NOT ] REGEXP pattern [ ESCAPE escape-expression ]
```

Parameters

- **expression** The string to be searched.

- **pattern** The regular expression to search for within expression.

- **escape-expression** The escape character to be used in the match. The default is the backslash character (\).

Remarks

The REGEXP search condition matches a whole string, not a substring. To match on a substring with the string, enclose the string in wildcards that match the rest of the string (. *pattern.*). For example, SELECT ... WHERE Description REGEXP 'car' matches only car, not sportscar. However, SELECT ... WHERE Description REGEXP '.*car' matches car, sportscar, and any string.
that ends with car. Alternatively, you can rewrite your query to make use the REGEXP_SUBSTR function, which is designed to search for substrings within a string.

When matching against only a sub-character class, you must include the outer square brackets and the square brackets for the sub-character class. For example, \texttt{expression \ REGEXP \ '[[:digit:]]'}. 

**Database collation and matching**

REGEXP only matches a literal in a pattern if it is the exact same character (that is, they have the same code point value). Ranges in character classes (for example, \texttt{[A-F]} ) only match characters that code point values greater than or equal to the code point value of the first character in the range (A) and less than or equal to the code point value of the second character in the range (F).

Comparisons are performed character by character, unlike the equivalence (\texttt{=} ) operator and other operators where the comparison is done string by string. For example, when a comparison is done in a UCA collation (CHAR or NCHAR with the collation set to UCA), \texttt{'Æ'='AE'} is true, but \texttt{'Æ' \ REGEXP 'Æ'} is false.

**National character (NCHAR) support**

REGEXP search conditions can be used to compare CHAR and NCHAR strings. In this case, character set conversion is performed so that the comparison is done using a common data type. Then, a code point by code point comparison is performed.

You can specify \texttt{expression} or \texttt{pattern} as an NCHAR string literal by prefixing the quoted value with \texttt{N} (for example, \texttt{expression \ REGEXP \ N'pattern'}). You can also use the \texttt{CAST} function to cast the pattern to CHAR or NCHAR (for example, \texttt{expression \ REGEXP \ CAST(pattern \ AS \ datatype)}).

**See also**

- “Comparisons between CHAR and NCHAR” on page 132
- “String literals” on page 8
- “CAST function [Data type conversion]” on page 174
- “Regular expressions overview” on page 26
- “SIMILAR TO search condition” on page 53
- “LIKE search condition” on page 47
- “REGEXP_SUBSTR function [String]” on page 329
- “LIKE, REGEXP, and SIMILAR TO search conditions” on page 45
- “Regular expressions: Special sub-character classes” on page 30

**Standards and compatibility**

- **SQL/2008**  The REGEXP search condition is a vendor extension, but is roughly compatible with the LIKE_REGEX search condition of the SQL/2008 standard, which is SQL language feature F841.

SQL Anywhere supports ANSI SQL/2008 feature F281, which permits the pattern and escape-expressions to be arbitrary expressions evaluated at execution time. Feature F281 also permits \texttt{expression} to be an expression more complex than a simple column reference.
SQL Anywhere supports ANSI SQL/2008 feature T042, which permits REGEXP search conditions to reference string-expressions that are LONG VARCHAR values.

REGEXP search conditions that specify NCHAR string expressions or patterns is feature F421 of the ANSI SQL/2008 standard.

**SIMILAR TO search condition**

Matches a pattern against a string.

**Syntax**

\[expression [ NOT ] SIMILAR TO pattern [ ESCAPE escape-expression ]\]

**Parameters**

- expression The expression to be searched.
- pattern The regular expression to search for within expression.
- escape-expression The escape character to use in the match. The default escape character is the null character, which can be specified in a string literal as 'x00'.

<table>
<thead>
<tr>
<th>Regular expression syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\x</td>
<td>Match anything that compares equal to x, where the escape character is assumed to be the backslash character (). For example, [ matches '['.</td>
</tr>
<tr>
<td>x</td>
<td>Any character (other than a meta-character) matches itself. For example, A matches 'A'.</td>
</tr>
</tbody>
</table>

**Remarks**

To match a substring with the string, use the percentage sign wildcard (%expression). For example, SELECT ... WHERE Description SIMILAR TO 'car' matches only car, not sportscar. However, SELECT ... WHERE Description SIMILAR TO '%car' matches car, sportscar, and any string that ends with car.

When matching against only a sub-character class, you must include the outer square brackets, and the square brackets for the sub-character class. For example, expression SIMILAR TO '[[[:digit:]]]'.

Comparisons are performed character by character, unlike the equivalence (=) operator and other operators where the comparison is done string by string. For example, when a comparison is done in a UCA collation (CHAR or NCHAR with the collation set to UCA), 'Æ'='Æ' is true, but 'Æ' SIMILAR TO 'Æ' is false.

For a character-by-character comparison to match, each single character in the expression being searched must match a single character or a wildcard in the SIMILAR TO pattern.
Database collation and matching

SIMILAR TO use the collation to determine character equivalence and evaluate character class ranges. For example, if the database is case- and accent-insensitive, matches are case- and accent-insensitive. Ranges are also evaluated using the collation sort order.

National character (NCHAR) support

SIMILAR TO search conditions can be used to compare CHAR and NCHAR strings. In this case, character set conversion is performed so that the comparison is done using a common data type. Then, a character-by-character comparison is performed.

You can specify expression or pattern as an NCHAR string literal by prefixing the quoted value with N (for example, expression SIMILAR TO N'pattern'). You can also use the CAST function to cast the pattern to CHAR or NCHAR (for example, expression SIMILAR TO CAST(pattern AS datatype)).

See also

- “Regular expressions overview” on page 26
- “Comparisons between CHAR and NCHAR” on page 132
- “String literals” on page 8
- “CAST function [Data type conversion]” on page 174
- “REGEXP search condition” on page 51
- “LIKE search condition” on page 47
- “REGEXP_SUBSTR function [String]” on page 329
- “Regular expressions: Special sub-character classes” on page 30
- “LIKE, REGEXP, and SIMILAR TO search conditions” on page 45

Standards and compatibility

- SQL/2008 The SIMILAR TO predicate is optional SQL language feature T141 of the SQL/2008 standard.

IN search condition

Syntax

expression [ NOT ] IN { ( query-expression ) | ( expression-list ) }  

Remarks

An IN search condition compares expression with the set of values returned by query-expression or the set of values specified in expression-list. Without the NOT keyword, the IN search condition evaluates according to the following rules:

- TRUE if expression is not NULL and equals at least one of the values.

- UNKNOWN if expression is NULL and the values list is not empty, or if at least one of the values is NULL and expression does not equal any of the other values.
• FALSE if `expression` is NULL and `query-expression` returns no values; or if `expression` is not NULL, none of the values are NULL, and `expression` does not equal any of the values.

The NOT keyword interchanges TRUE and FALSE.

The search condition `expression IN ( expression-list )` is equivalent to `expression = ANY ( expression-list )`.

The search condition `expression NOT IN ( expression-list )` is equivalent to `expression <> ALL ( expression-list )`.

The expressions in an `expression-list` can be a literal, variable, host variable, or a query expression whose result is a single row and a single column.

**Standards and compatibility**

• SQL/2008 Core feature.

**CONTAINS search condition**

**Syntax**

```sql
CONTAINS ( column-name [, ...], contains-query-string )
contains-query-string:
simple-expression
| or-expression

simple-expression:
primary-expression
| and-expression

or-expression:
simple-expression { OR | | } contains-query-string

primary-expression:
basic-expression
| FUZZY "fuzzy-expression"
| and-not-expression

and-expression:
primary-expression [ AND | & ] simple-expression

and-not-expression:
primary-expression [ AND | & ] { NOT | - } basic-expression

basic-expression:
term
| phrase
| ( contains-query-string )
| near-expression
| before-expression
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 55
fuzzy-expression :
term
| fuzzy-expression term

term :
simple-term
| prefix-term

prefix-term :
simple-term*

phrase :
" phrase-string "

near-expression :
term NEAR [ [ min-distance ], max-distance ] term
| term { NEAR | ~ } term

before-expressions :
term BEFORE [ [ min-distance ] max-distance ] term
| term BEFORE term

phrase-string :
term
| phrase-string term

simple-term : a string separated by whitespace and special characters that represents a single indexed term (word) to search for

distance : a positive integer

Parameters

- **and-expression**  Specifies that both primary-expression and simple-expression must be found in the text index.

  By default, if no operator is specified between terms or expressions, an and-expression is assumed. For example, 'a b' is interpreted as 'a AND b'.

  An ampersand (&) can be used instead of AND, and can abut the expressions or terms on either side (for example, 'a & b').

- **and-not-expression**  Specifies that primary-expression must be present in the text index, but that basic-expression must not be found in the text index. This is also known as a negation.

  If you use a hyphen for negation, the hyphen must have a space to the left of it, and must adjoin the term to the right; otherwise, the hyphen is not interpreted as a negation. For example, 'a ~ b' is equivalent to 'a AND NOT b'; whereas for 'a - b', the hyphen is ignored and the string is equivalent to 'a AND b'. 'a~b' is equivalent to the phrase '"a b"'.

- **or-expression**  Specifies that at least one of simple-expression or contains-query-string must be present in the text index. For example, 'a | b' is interpreted as 'a OR b'.
fuzzy-expression  Finds terms that are similar to what you specify. Fuzzy matching is only supported on NGRAM text indexes.

near-expression  Searches for terms that are near each other. This is also known as a proximity search. For example, 'b NEAR[5] c' searches for instances of b and c that are five or fewer terms away from each other. The order of terms is not significant; 'b NEAR c' is equivalent to 'c NEAR b'.

If the maximum distance is not specified, the default distance is 10. If the minimum distance is not specified, the default is 1.

The query 'apple NEAR[2, 10] tree' matches the following documents:

'apple grows on the tree'
'apple and tree'
'tree and apple'

However, the query does not match the following documents:

'apple tree'
'tree apple'

You can specify a tilde (~) instead of NEAR. Using a tilde is equivalent to specifying NEAR without a distance, so a default of maximum 10 terms and minimum 1 term is applied. You cannot specify a maximum or minimum distance if you specify a tilde; it is always 10 terms.

NEAR expressions cannot be chained together (for example, 'a NEAR[1] b NEAR[1] c' is invalid).

before-expression  Use before-expression to search for a term that is before another term. This is also known as a proximity search. The arguments in the matching text must be found in the same order as they are specified in the CONTAINS query string. For example, 'apple BEFORE[2, 10] tree' matches the following documents:

'apple grows on the tree'
'apple and tree'

However, the query does not match the following documents:

'tree and apple'
'apple tree'
'tree apple'

The following queries are equivalent:

'apple BEFORE tree'
'apple BEFORE[10] tree'
'apple BEFORE[1, 10] tree'

If the maximum distance is not specified, the default distance is 10. If the minimum distance is not specified, the default is 1.
• **prefix-term**  Searches for terms that start with the specified prefix. For example, 'datab*' searches for any term beginning with datab. This is also known as a **prefix search**. In a prefix search, matching is performed for the portion of the term to the left of the asterisk.

**Remarks**

The CONTAINS search condition takes a column list and `contains-query-string` as arguments. It can be used anywhere that a search condition (also referred to as predicate) can be specified, and returns TRUE or FALSE. The `contains-query-string` must be a constant string or a variable, with a value that is known at query time. The `contains-query-string` cannot be NULL, an empty string, or exceed 300 valid terms. A valid term is a term that is within the permitted term length and is not included in the STOPLIST. An error is returned when the `contains-query-string` exceeds 300 valid terms.

If the text configuration settings cause all of the terms in the `contains-query-string` to be dropped, the result of the CONTAINS search condition is FALSE.

For more information about how the `contains-query-string` is interpreted, see “Example text configuration objects” [SQL Anywhere Server - SQL Usage].

If multiple columns are specified, then they must all refer to a single base table; a text index cannot span multiple base tables. The base table can be referenced directly in the FROM clause, or it can be used in a view or derived table.

The following warnings apply to the use of non-alphanumeric characters in query strings:

• An asterisk in the middle of a term returns an error.

• Do not use non-alphanumerics (including special characters) in `fuzzy-expression` because they are treated as whitespace and serve as term breakers.

• If possible, do not include non-alphanumeric characters that are not special characters in your query string. Any non-alphanumeric character that is not a special character causes the term containing it to be treated as a phrase, breaking the term at the location of the character. For example, 'things we've done' is interpreted as 'things "we ve" done'.

Within phrases, the asterisk is the only special character that continues to be interpreted as a special character. All other special characters within phrases are treated as whitespace and serve as term breakers.

Interpretation of `contains-query-string` takes place in two main steps:

• **Step 1: Interpreting operators and precedence**  During this step, keywords are interpreted as operators, and rules of precedence are applied.

• **Step 2: Applying text configuration object settings**  During this step, the text configuration object settings are applied to terms. For example, on an NGRAM text index, terms are broken down into their n-gram representation. During this step, the query terms that exceed the term length settings, or that are in the stoplist, are dropped.

**Operator precedence in a CONTAINS search condition**

During query evaluation, expressions are evaluated using the following order of precedence:
1. FUZZY, NEAR
2. AND NOT
3. AND
4. OR

Treatment of BEFORE as a keyword

SQL Anywhere does not currently support the BEFORE keyword as an operator. For example, if you specify CONTAINS(column-name, 'a before b'), an error is returned. Construct your query using the NEAR keyword instead.

You can search for the word before, providing it is part of a phrase query. For example, CONTAINS(column-name, '"a before b"'). This query searches for the phrase "a before b".

Allowed syntax for asterisk (*)

The asterisk is used for prefix searching. An asterisk can occur at the end of the query string, or be followed by a space, ampersand, vertical bar, closing bracket, or closing quotation mark. Any other usage of asterisk returns an error.

The following table shows allowable asterisk usage:

<table>
<thead>
<tr>
<th>Query string</th>
<th>Equivalent to</th>
<th>Interpreted as</th>
</tr>
</thead>
<tbody>
<tr>
<td>'th*'</td>
<td></td>
<td>Find any term beginning with th.</td>
</tr>
<tr>
<td>'th*&amp;best'</td>
<td>'th* AND best' and 'th* best'</td>
<td>Find any term beginning with th, and the term best.</td>
</tr>
<tr>
<td>'th*</td>
<td>best'</td>
<td>'th* OR best'</td>
</tr>
<tr>
<td>'very&amp;(best</td>
<td>th*)'</td>
<td>'very AND (best OR th*)'</td>
</tr>
<tr>
<td>'&quot;fast auto*&quot;'</td>
<td></td>
<td>Find the term fast, immediately followed by a term beginning with auto.</td>
</tr>
<tr>
<td>'&quot;auto* price&quot;'</td>
<td></td>
<td>Find a term beginning with auto, immediately followed by the term price.</td>
</tr>
</tbody>
</table>

Note
Interpretation of query strings containing asterisks can vary depending on the text configuration object settings.
Allowed syntax for hyphen (-)

The hyphen can be used for term or expression **negation** and is equivalent to NOT. Whether a hyphen is interpreted as a negation depends on its location in the query string. For example, when a hyphen immediately precedes a term or expression, it is interpreted as a negation. If the hyphen is embedded within a term, it is interpreted as a hyphen.

A hyphen used for negation must be preceded by a whitespace and followed immediately by an expression.

When used in a phrase of a fuzzy expression, the hyphen is treated as whitespace and used as a term breaker.

The following table shows the allowed syntax for hyphen:

<table>
<thead>
<tr>
<th>Query string</th>
<th>Equivalent to:</th>
<th>Interpreted as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>'the -best'</td>
<td>'the AND NOT best', 'the AND -best', 'the &amp; -best', 'the NOT best'</td>
<td>Find the term the, and not the term best.</td>
</tr>
<tr>
<td>'the -(very best)'</td>
<td>'the AND NOT (very AND best)'</td>
<td>Find the term the, and not the terms very and best.</td>
</tr>
<tr>
<td>'the -&quot;very best&quot;'</td>
<td>'the AND NOT &quot;very best&quot;'</td>
<td>Find the term the, and not the phrase very best.</td>
</tr>
<tr>
<td>'alpha-numer-ics'</td>
<td>&quot;alpha numerics&quot;</td>
<td>Find the term alpha, immediately followed by the term numerics.</td>
</tr>
<tr>
<td>'wild - west'</td>
<td>'wild west', and 'wild AND west'</td>
<td>Find the term wild, and the term west.</td>
</tr>
</tbody>
</table>

Allowed syntax for special characters

The following table shows the allowed syntax for all special characters except asterisk and hyphen.

These characters are not considered special characters if they are found in a phrase, and are dropped.

**Note**

The same restrictions with regards to specifying string literals also apply to the query string. For example, apostrophes must be escaped, and so on.
<table>
<thead>
<tr>
<th>Character or syntax</th>
<th>Usage examples and remarks</th>
</tr>
</thead>
</table>
| **ampersand (&)**   | The ampersand is equivalent to AND and can be specified as follows:  
  - 'a & b'  
  - 'a &b'  
  - 'a&b'  
  - 'a& b' |
| **vertical bar ()** | The vertical bar is equivalent to OR and can be specified as follows:  
  - 'a|b'  
  - 'a |b'  
  - 'a | b'  
  - 'a| b' |
| **double quotes ()**| Double quotes are used to contain a sequence of terms where order and relative distance are important. For example, in the query string 'learn "full text search"', **full text search** is a phrase. In this example, learn can come before or after the phrase, or exist in another column (if the text index is built on more than one column), but the exact phrase must be found in a single column. |
| **parentheses ()**  | Parentheses are used to specify the order of evaluation of expressions if different from the default order. For example 'a AND (b|c)' is interpreted as a, and b or c. |
| **tilde (~)**       | The tilde is equivalent to NEAR[10]. The query string 'full~text' is equivalent to 'full NEAR text', and is interpreted as: the term full within ten terms of the term text.  
  You cannot specify a distance with the tilde. |
| **square brackets []** | Square brackets are used with the keyword NEAR to contain distance. Other uses of square brackets return an error. |

**Standards and compatibility**
- **SQL/2008**  The CONTAINS predicate is a vendor extension.
EXISTS search condition

Syntax

```sql
EXISTS ( subquery )
```

Remarks

The EXISTS search condition is TRUE if the subquery result contains at least one row, and FALSE if the subquery result does not contain any rows. The EXISTS search condition cannot be UNKNOWN.

Standards and compatibility

- SQL/2008 Core feature.

IS NULL and IS NOT NULL search conditions

Syntax

```sql
expression IS [ NOT ] NULL
```

Remarks

Without the NOT keyword, the IS NULL search condition is TRUE if the expression is the NULL value, and FALSE otherwise. The NOT keyword reverses the meaning of the search condition.

Standards and compatibility

- SQL/2008 Core feature.
Truth value search conditions

Syntax

IS [ NOT ] truth-value

Remarks

Without the NOT keyword, the search condition is TRUE if the condition evaluates to the supplied truth-value, which must be one of TRUE, FALSE, or UNKNOWN. Otherwise, the value is FALSE. The NOT keyword reverses the meaning of the search condition, but leaves UNKNOWN unchanged.

Standards and compatibility

- SQL/2008  Truth value search conditions comprise optional SQL language feature F571 of the SQL/2008 standard.

Three-valued logic

The following tables display how the AND, OR, NOT, and IS logical operators of SQL work in three-valued logic.

AND operator

<table>
<thead>
<tr>
<th>AND</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

OR operator

<table>
<thead>
<tr>
<th>OR</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

NOT operator

<table>
<thead>
<tr>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>
### IS operator

<table>
<thead>
<tr>
<th>IS</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

### Standards and compatibility
- **SQL/2008** Core feature. Truth value tests, such as IS UNKNOWN, comprise SQL language feature F571.

### See also
- “NULL special value” on page 71

### Explicit selectivity estimates

SQL Anywhere uses statistical information to determine the most efficient strategy for executing each statement. SQL Anywhere automatically gathers and updates these statistics. These statistics are stored permanently in the database in the system table ISYSCOLSTAT. Statistics gathered while processing one statement are available when searching for efficient ways to execute subsequent statements.

Occasionally, the statistics may become inaccurate or relevant statistics may be unavailable. This condition is most likely to arise when few queries have been executed since a large amount of data was added, updated, or deleted. In this situation, you may want to execute a CREATE STATISTICS statement.

If there are problems with a particular execution plan, you can use optimizer hints to require that a particular index be used.

In unusual circumstances, however, these measures may prove ineffective. In such cases, you can sometimes improve performance by supplying explicit selectivity estimates.

For each table in a potential execution plan, the optimizer must estimate the number of rows that will be part of the result set. If you know that a condition has a success rate that differs from the optimizer's estimate, you can explicitly supply a user estimate in the search condition.

The estimate is a percentage. It can be a positive integer or can contain fractional values.

### Caution

Whenever possible, avoid supplying explicit estimates in statements that are to be used on an ongoing basis. Should the data change, the explicit estimate may become inaccurate and may force the optimizer to select poor plans. If you do use explicit selectivity estimates, ensure that the number is accurate. Do not, for example, supply values of 0% or 100% to force the use of an index.
You can disable user estimates by setting the database option user_estimates to Off. The default value for user_estimates is Override-Magic, which means that user-supplied selectivity estimates are used only when the optimizer would use a MAGIC (default) selectivity value for the condition. The optimizer uses MAGIC values as a last resort when it is unable to accurately predict the selectivity of a predicate.

Examples

The following query provides an estimate that one percent of the ShipDate values are later than 2001/06/30:

```sql
SELECT  ShipDate
FROM  GROUPO.SalesOrderItems
WHERE ( ShipDate > '2001/06/30', 1 )
ORDER BY ShipDate DESC;
```

The following query estimates that half a percent of the rows satisfy the condition:

```sql
SELECT *
FROM GROUPO.Customers c, GROUPO.SalesOrders o
WHERE (c.ID = o.CustomerID, 0.5);
```

Fractional values enable more accurate user estimates for joins and large tables.

Standards and compatibility

- SQL/2008  Vendor extension.

See also

- “CREATE STATISTICS statement” on page 682
- “FROM clause” on page 810
- “user_estimates option” [SQL Anywhere Server - Database Administration]
- “Optimizer estimates and statistics” [SQL Anywhere Server - SQL Usage]

Special values

Special values can be used in expressions, and as column defaults when creating tables.

While some special values can be queried, some can only be used as default values for columns. For example, LAST USER, TIMESTAMP and UTC TIMESTAMP can only be used as default values.

**CURRENT DATABASE special value**

CURRENT DATABASE returns the name of the current database.

Data type

string

See also

- “Expressions” on page 21
Standards and compatibility
- SQL/2008 Vendor extension.

CURRENT DATE special value
CURRENT DATE returns the present year, month, and day.

Data type
DATE

See also
- “DATE data type” on page 115
- “DATE function [Date and time]” on page 203
- “DATETIME data type” on page 116
- “DATETIME function [Date and time]” on page 209
- “DATETIMEOFFSET data type” on page 117
- “Expressions” on page 21
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120

Standards and compatibility
- SQL/2008 Vendor extension. In the ANSI SQL/2008 standard, the special register that defines the present date is called CURRENT_DATE. SQL Anywhere does not support CURRENT_DATE.

CURRENT PUBLISHER special value
CURRENT PUBLISHER returns a string that contains the publisher user ID of the database for SQL Remote replications.

Data type
string

Remarks
The publisher is set using the PUBLIC.db_publisher option, or by using the GRANT PUBLISH and REVOKE PUBLISH statements.

CURRENT PUBLISHER can be used as a default value in columns with character data types.
See also
- “db_publisher option” [SQL Anywhere Server - Database Administration]
- “GRANT PUBLISH statement [SQL Remote]” on page 839
- “REVOKE PUBLISH statement [SQL Remote]” on page 943
- “Viewing the publisher” [SQL Remote]

Standards and compatibility
- SQL/2008 Vendor extension.

**CURRENT REMOTE USER special value**

If the current connection belongs to the receive phase of SQL Remote, then CURRENT REMOTE USER returns the user ID of the remote user that created the messages that are currently being applied on this connection. In all other circumstances, CURRENT REMOTE USER is a NULL value.

**Data type**
string

**Remarks**
The CURRENT REMOTE USER special value is set by the receive phase of SQL Remote when it is applying messages to the database. The CURRENT REMOTE USER special value is most useful in triggers to determine whether the operations being applied are being applied by the receive phase of SQL Remote, and if they are, which remote user generated the operations being applied.

See also
- “Using the CURRENT REMOTE USER special value” [SQL Remote]
- “-t option, SQL Remote Message Agent utility (dbremote)” [SQL Remote]

Standards and compatibility
Vendor extension.

**CURRENT TIME special value**

CURRENT TIME returns the present hour, minute, second, and fraction of a second.

**Data type**
TIME

**Remarks**
The fraction of a second is stored to 6 decimal places. The accuracy of the present time is limited by the accuracy of the system clock.
Standards and compatibility

- **SQL/2008**  Vendor extension. In the ANSI SQL/2008 standard, the special register that defines the present time is called CURRENT_TIME. SQL Anywhere does not support CURRENT_TIME.

**CURRENT TIMESTAMP special value**

CURRENT_TIMESTAMP combines CURRENT_DATE and CURRENT_TIME to form a TIMESTAMP value containing the year, month, day, hour, minute, second and fraction of a second.

**Data type**

TIMESTAMP

**Remarks**

The fraction of a second is stored to 6 decimal places. The accuracy of the present time is limited by the accuracy of the system clock.

Unlike DEFAULT_TIMESTAMP, columns declared with DEFAULT CURRENT_TIMESTAMP do not necessarily contain unique values. If uniqueness is required, consider using DEFAULT_TIMESTAMP instead.

The information CURRENT_TIMESTAMP returns is equivalent to the information returned by the GETDATE and NOW functions.

CURRENT_TIMESTAMP is equivalent to CURRENT_TIMESTAMP.
Note
The main difference between DEFAULT CURRENT TIMESTAMP and DEFAULT TIMESTAMP is that DEFAULT CURRENT TIMESTAMP is set only at INSERT, while DEFAULT TIMESTAMP is set at both INSERT and UPDATE.

See also
- “Expressions” on page 21
- “CURRENT TIME special value” on page 67
- “CURRENT UTC TIMESTAMP special value” on page 70
- “DATE data type” on page 115
- “DATE function [Date and time]” on page 203
- “DATETIME data type” on page 116
- “DATETIME function [Date and time]” on page 209
- “DATETIMEOFFSET data type” on page 117
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP data type” on page 121
- “TIMESTAMP special value” on page 76
- “UTC TIMESTAMP special value” on page 78

Standards and compatibility
- SQL/2008 Vendor extension. In the SQL/2008 standard, the special register that defines the present timestamp is called CURRENT_TIMESTAMP.

CURRENT USER special value

CURRENT USER contains the user ID of the current connection.

Data type

string

Remarks

CURRENT USER can be used as a default value in columns with character data types.

On UPDATE, columns with the CURRENT USER default are not changed unless explicitly updated. The LAST USER default may be used to track updates by users.

CURRENT_USER is equivalent to CURRENT USER.
Standards and compatibility

● **SQL/2008**  Vendor extension. In the SQL/2008 standard, the special register that defines the current user is called CURRENT_USER.

**CURRENT UTC TIMESTAMP special value**

CURRENT UTC TIMESTAMP returns the Coordinated Universal Time (UTC) containing the year, month, day, hour, minute, second, fraction of a second, and time zone.

**Data type**

TIMESTAMP WITH TIME ZONE

**Remarks**

This feature allows data to be entered with a consistent time reference, regardless of the time zone in which the data was entered.

See also

● “Expressions” on page 21
● “CURRENT TIME special value” on page 67
● “CURRENT TIMESTAMP special value” on page 68
● “DATE data type” on page 115
● “DATE function [Date and time]” on page 203
● “DATETIME data type” on page 116
● “DATETIME function [Date and time]” on page 209
● “DATETIMEOFFSET data type” on page 117
● “GETDATE function [Date and time]” on page 251
● “ISDATE function [Data type conversion]” on page 275
● “NOW function [Date and time]” on page 311
● “SMALLDATETIME data type” on page 119
● “TIME data type” on page 120
● “TIMESTAMP data type” on page 121
● “TIMESTAMP special value” on page 76
● “UTC TIMESTAMP special value” on page 78

Standards and compatibility

● **SQL/2008**  Vendor extension. The TIMESTAMP WITH TIME ZONE data type is optional SQL language feature F411 in the SQL/2008 standard.
LAST USER special value

LAST USER is the user ID of the user who last modified the row.

Data type

String

Remarks

LAST USER can be used as a default value in columns with character data types.

On INSERT, this constant has the same effect as CURRENT USER.

On UPDATE, if a column with a default value of LAST USER is not explicitly modified, it is changed to the name of the current user.

When combined with the DEFAULT TIMESTAMP, a default value of LAST USER can be used to record (in separate columns) both the user and the date and time of day a row was last changed.

See also

● “CREATE TABLE statement” on page 690
● “CURRENT USER special value” on page 69
● “CURRENT TIMESTAMP special value” on page 68
● “USER special value” on page 77

Standards and compatibility

● SQL/2008 Vendor extension.

NULL special value

The NULL value specifies a value that is unknown or not applicable.

Syntax

NULL

Remarks

NULL is a special value that is different from any valid value for any data type. However, the NULL value is a legal value in any data type. NULL is used to represent missing or inapplicable information. There are two separate and distinct cases where NULL is used:

<table>
<thead>
<tr>
<th>Situation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>missing</td>
<td>The field does have a value, but that value is unknown.</td>
</tr>
<tr>
<td>inapplicable</td>
<td>The field does not apply for this particular row.</td>
</tr>
</tbody>
</table>

SQL allows columns to be created with the NOT NULL restriction. Those particular columns cannot contain NULL.
The NULL value introduces the concept of three valued logic to SQL. The NULL value compared using any comparison operator with any value (including the NULL value) is "UNKNOWN." The only search condition that returns TRUE is the IS NULL predicate. In SQL, rows are selected only if the search condition in the WHERE clause evaluates to TRUE; rows that evaluate to UNKNOWN or FALSE are not selected.

Column space utilization for NULL values is 1 bit per column and space is allocated in multiples of 8 bits. The NULL bit usage is fixed based on the number of columns in the table that allow NULL values.

The IS [ NOT ] truth-value clause, where truth-value is one of TRUE, FALSE or UNKNOWN can be used to select rows where the NULL value is involved.

In the following examples, the column Salary contains NULL.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Truth value</th>
<th>Selected?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salary = NULL</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>Salary &lt;&gt; NULL</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>NOT (Salary = NULL)</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>NOT (Salary &lt;&gt; NULL)</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>Salary = 1000</td>
<td>UNKNOWN</td>
<td>NO</td>
</tr>
<tr>
<td>Salary IS NULL</td>
<td>TRUE</td>
<td>YES</td>
</tr>
<tr>
<td>Salary IS NOT NULL</td>
<td>FALSE</td>
<td>NO</td>
</tr>
<tr>
<td>Salary = expression IS UNKNOWN</td>
<td>TRUE</td>
<td>YES</td>
</tr>
</tbody>
</table>

The same rules apply when comparing columns from two different tables. Therefore, joining two tables together does not select rows where any of the columns compared contain the NULL value.

NULL also has an interesting property when used in numeric expressions. The result of any numeric expression involving the NULL value is NULL. If NULL is added to a number, then the result is NULL—not a number. To treat NULL as 0, use the ISNULL( expression, 0 ) function.

Many common errors in formulating SQL queries are caused by the behavior of NULL. You have to be careful to avoid these problem areas.

**Set operators and DISTINCT clause**

In SQL, comparisons to NULL within search conditions yield UNKNOWN as the result. However, when determining whether or not two rows are duplicates of each other, SQL treats NULL as equivalent to NULL. These semantics apply to the set operators (UNION, INTERSECT, EXCEPT), GROUP BY, PARTITION within a WINDOW clause, and SELECT DISTINCT.

For example, if a column called redundant contained NULL for every row in a table T1, then the following statement would return a single row:
SELECT DISTINCT redundant FROM T1;

Prerequisites

Must be connected to the database.

Side effects

None.

Standards and compatibility

- **SQL/2008** Core feature.

- **Transact-SQL** In some contexts, Adaptive Server Enterprise treats comparisons to NULL values differently. If an expression is compared to a variable or NULL literal using equality or inequality, and if expression is a simple expression that refers to the column of a base table or view, then the comparison is performed using two-valued logic, with NULL = NULL yielding TRUE rather than UNKNOWN. The list of possible comparisons with these semantics, and their SQL/2008 equivalents, are as follows:

<table>
<thead>
<tr>
<th>Transact-SQL comparison</th>
<th>SQL/2008 equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression = NULL</td>
<td>expression IS NULL</td>
</tr>
<tr>
<td>expression != NULL</td>
<td>NOT (expression IS NULL)</td>
</tr>
<tr>
<td>expression = variable</td>
<td>expression = variable IS TRUE OR (expression IS NULL AND variable IS NULL)</td>
</tr>
<tr>
<td>expression != variable</td>
<td>expression != variable IS TRUE AND ( NOT expression IS NULL OR NOT variable IS NULL)</td>
</tr>
</tbody>
</table>

SQL Anywhere will implement these semantics to match Adaptive Server Enterprise behavior if the ansinull option is set to OFF. The ansinull option is set to OFF by default for Open Client and jConnect connections. To ensure SQL/2008 semantics, you can either reset the ansinull option to ON, or use an IS [NOT] NULL predicate instead of an equality comparison.

Unique indexes in SQL Anywhere can hold rows that hold NULL and are otherwise identical. Adaptive Server Enterprise does not permit such entries in unique indexes.

If you use jConnect, the tds_empty_string_is_null option controls whether empty strings are returned as NULL strings or as a string containing one blank character.

See also

- “tds_empty_string_is_null option” [SQL Anywhere Server - Database Administration]
- “Expressions” on page 21
- “ansinull option” [SQL Anywhere Server - Database Administration]
- “tds_empty_string_is_null option” [SQL Anywhere Server - Database Administration]
- “Search conditions” on page 40
Example

The following INSERT statement inserts a NULL into the date_returned column of the Borrowed_book table.

```sql
INSERT INTO Borrowed_book ( date_borrowed, date_returned, book )
VALUES ( CURRENT DATE, NULL, '1234' );
```

SQLCODE special value

SQLCODE indicates the disposition of the most recently executed SQL statement.

Data type

Signed INTEGER

Remarks

The database server sets a SQLSTATE and SQLCODE for each SQL statement it executes. SQLCODEs are product-specific (for example, MobiLink has its own SQLCODEs), and can be used to learn additional information about the SQLSTATE. For example, positive values other than 100 indicate product-specific warning conditions. Negative values indicate product-specific exception conditions. The value 100 indicates "no data" (for example, at the end of a result set fetched via a cursor).

SQLSTATE and SQLCODE are related in that each SQLCODE corresponds to a SQLSTATE, and each SQLSTATE can correspond to one or more SQLCODEs.

To return the error condition associated with a SQLCODE, you can use the ERRORMSG function.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLSTATE is the preferred status indicator for the outcome of a SQL statement.</td>
</tr>
</tbody>
</table>

See also

- “SQLSTATE special value” on page 74
- “SQL Anywhere error messages sorted by SQLCODE” [Error Messages]
- “Expressions” on page 21
- “ERRORMSG function [Miscellaneous]” on page 235

Standards and compatibility

- **SQL/2008** SQLCODE was deprecated in the ANSI SQL/1992 standard, and was eliminated entirely from SQL/1999. SQLCODE values continue to be maintained in SQL Anywhere for backward compatibility for applications. SQLSTATE is the preferred status indicator.

SQLSTATE special value

SQLSTATE indicates whether the most recently executed SQL statement resulted in a success, error, or warning condition.
**Data type**

String

**Remarks**

The database server sets a SQLSTATE and SQLCODE for each SQL statement it executes. A SQLSTATE is a string that indicates whether the most recently executed SQL statement resulted in a success, warning, or error condition.

Each SQLSTATE represents errors that are common to all platforms, and usually contain non-product-specific wording. The format of a SQLSTATE value is a two-character class value, followed by a three-character subclass value. Guidelines for SQLSTATE conformance with regard to class and subclass values are outlined in the ISO/ANSI SQL standard.

SQL Anywhere conforms to the ISO/ANSI SQLSTATE conventions with the following additions and exceptions:

<table>
<thead>
<tr>
<th>Class and subclass</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>01WC.x</td>
<td>Warnings related to character set conversion</td>
</tr>
<tr>
<td>38.xxx</td>
<td>External function exception</td>
</tr>
<tr>
<td>42Xxx</td>
<td>Syntax error: expressions</td>
</tr>
<tr>
<td>42Rxx</td>
<td>Syntax error: referential integrity (for example, attempt to create second primary key)</td>
</tr>
<tr>
<td>42Wxx</td>
<td>Syntax error: generic</td>
</tr>
<tr>
<td>42Uxx</td>
<td>Syntax error: duplicate, undefined, or ambiguous object reference</td>
</tr>
<tr>
<td>42Zxx</td>
<td>Access violation</td>
</tr>
<tr>
<td>54Wxx</td>
<td>Product limit exceeded</td>
</tr>
<tr>
<td>55Wxx</td>
<td>Object not in required state for operation to succeed</td>
</tr>
<tr>
<td>57xxx</td>
<td>Resource not available or operator intervention</td>
</tr>
<tr>
<td>5Rxxx</td>
<td>SQL Remote errors</td>
</tr>
<tr>
<td>WB.xxx</td>
<td>Online backup errors</td>
</tr>
<tr>
<td>WL.xxx</td>
<td>Internal database errors</td>
</tr>
<tr>
<td>WP.xxx</td>
<td>Errors in procedures, variables, and so on</td>
</tr>
<tr>
<td>WL.xxx</td>
<td>Errors loading and/or unloading</td>
</tr>
</tbody>
</table>
Class and subclass | Condition |
---|---|
WW\_xxx | Miscellaneous SQL Anywhere-specific errors/warnings (including system failures) |
WO\_xxx | Remote data access feature-related errors |
WJ\_xxx | JCS and JDBC related errors |
WC\_xxx | Character translation errors |
WX\_xxx | XML-related errors |
WT\_xxx | Text-related errors |

The successful completion class is '00\_xxx' (for example, '00000').

SQLSTATE and SQLCODE are related in that each SQLCODE corresponds to a SQLSTATE, and each SQLSTATE can correspond to one or more SQLCODEs.

To return the error condition associated with a SQLSTATE, you can use the ERRORMSG function.

See also
- “ERRORMSG function [Miscellaneous]” on page 235
- “SQLCODE special value” on page 74
- “SQL Anywhere error messages sorted by SQLSTATE” [Error Messages]
- “Expressions” on page 21

Standards and compatibility
  - SQL/2008 SQLSTATE classes (the first two characters) beginning with the values '0'-'4', and 'A'-'H' are defined by the ANSI standard. Other classes are implementation-defined. Similarly, subclass values that begin with values '0'-'4', and 'A'-'H' are defined by the ANSI standard. Subclass values outside these ranges are implementation-defined.

### TIMESTAMP special value

The TIMESTAMP default value is used to record the local date and time of day when a row in a table was last modified.

**Data type**

TIMESTAMP

**Remarks**

The fraction of a second is stored to 6 decimal places. The accuracy of the present time is limited by the accuracy of the system clock.
When a column is declared with DEFAULT TIMESTAMP, a default value is provided for inserts, and the value is updated with the present date and time of day whenever the row is updated.

Columns declared with DEFAULT TIMESTAMP contain unique values so that applications can detect near-simultaneous updates to the same row. If the present timestamp value is the same as the last value, it is incremented by the value of the default_timestamp_increment option.

You can automatically truncate timestamp values in SQL Anywhere based on the default_timestamp_increment option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

The global variable @@dbts returns a TIMESTAMP value representing the last value generated for a column using DEFAULT TIMESTAMP.

**Note**
The main difference between DEFAULT TIMESTAMP and DEFAULT CURRENT TIMESTAMP is that DEFAULT CURRENT TIMESTAMP is set only at INSERT, while DEFAULT TIMESTAMP is set at both INSERT and UPDATE.

### See also

- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMP special value” on page 70
- “DATE data type” on page 115
- “DATE function [Date and time]” on page 203
- “DATETIME data type” on page 116
- “DATETIME function [Date and time]” on page 209
- “DATETIMEOFFSET data type” on page 117
- “default_timestamp_increment option” [SQL Anywhere Server - Database Administration]
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP data type” on page 121
- “truncate_timestamp_values option” [SQL Anywhere Server - Database Administration]
- “UTC TIMESTAMP special value” on page 78

### Standards and compatibility

- **SQL/2008**  Vendor extension.

### USER special value

USER contains the user ID of the current connection.
Data type
string

Remarks
USER can be used as a default value in columns with character data types. It is equivalent to CURRENT USER.

On UPDATE, columns with the USER default are not changed unless explicitly updated. Instead, the LAST USER default may be used to track updates by users.

See also
● “Expressions” on page 21
● “CURRENT USER special value” on page 69
● “LAST USER special value” on page 71

Standards and compatibility
● SQL/2008 Vendor extension.

UTC TIMESTAMPTION special value
The UTC TIMESTAMP default value is used to record the Coordinated Universal Time (UTC) when a row in a table was last modified.

Data type
TIMESTAMP WITH TIME ZONE

Remarks
The fraction of a second is stored to 6 decimal places. The accuracy of the present time is limited by the accuracy of the system clock.

When a column is declared with DEFAULT UTC TIMESTAMPTION, a default value is provided for inserts, and the value is updated with the present UTC date and time of day whenever the row is updated.

Columns declared with DEFAULT UTC TIMESTAMPTION contain unique values so that applications can detect near-simultaneous updates to the same row. If the present UTC timestamp value is the same as the last value, it is incremented by the value of the default_timestamp_increment option.

You can automatically truncate UTC TIMESTAMPTION values in SQL Anywhere with the default_timestamp_increment option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

Note
DEFAULT UTC TIMESTAMPTION is set at both INSERT and UPDATE and DEFAULT CURRENT UTC TIMESTAMPTION is set at INSERT.
See also

- “Expressions” on page 21
- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMP special value” on page 70
- “DATE data type” on page 115
- “DATE function [Date and time]” on page 203
- “DATETIME data type” on page 116
- “DATETIME function [Date and time]” on page 209
- “DATETIMEOFFSET data type” on page 117
- “default_timestamp_increment option” [SQL Anywhere Server - Database Administration]
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP data type” on page 121
- “TIMESTAMP special value” on page 76
- “TIMESTAMP WITH TIME ZONE data type” on page 123
- “truncate_timestamp_values option” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008  Vendor extension.

Variables

SQL Anywhere supports three levels of variables:

- **Local variables**  These are defined inside a compound statement in a procedure or batch using the DECLARE statement. They exist only inside the compound statement.

- **Connection-level variables**  These are defined with a CREATE VARIABLE statement. They belong to the current connection, and disappear when you disconnect from the database or when you use the DROP VARIABLE statement.

- **Global variables**  These are system-supplied variables that have system-supplied values. All global variables have names beginning with two @ signs. For example, the global variable @@version has a value that is the current version number of the database server. Users cannot define global variables.

Local and connection-level variables are declared by the user, and can be used in procedures or in batches of SQL statements to hold information. Global variables are system-supplied variables that provide system-supplied values.

See also

- “TIMESTAMP data type” on page 121
- “CREATE VARIABLE statement” on page 722
Standards and compatibility

- **SQL/2008** Variables declared within SQL stored procedures or functions using the DECLARE statement is supported in the ANSI SQL/2008 standard as SQL language feature P002, "Computational completeness". CREATE VARIABLE, DROP VARIABLE, and global variables are all vendor extensions.

Local variables

Local variables are declared using the DECLARE statement, which can be used only within a compound statement (that is, bracketed by the BEGIN and END keywords). Only one variable can be declared for each DECLARE statement in SQL Anywhere.

If the DECLARE is executed within a compound statement, the scope is limited to the compound statement.

The variable is initially set as NULL. The value of the variable can be set using the SET statement, or can be assigned using a SELECT statement with an INTO clause.

The syntax of the DECLARE statement is as follows:

```
DECLARE variable-name data-type
```

Local variables can be passed as arguments to procedures, as long as the procedure is called from within the compound statement.

Examples

The following batch illustrates the use of local variables.

```
BEGIN
  DECLARE local_var INT;
  SET local_var = 10;
  MESSAGE 'local_var = ', local_var TO CLIENT;
END
```

Running this batch from Interactive SQL displays the message `local_var = 10` on the Messages tab of the Interactive SQL Results pane.

The variable local_var does not exist outside the compound statement in which it is declared. The following batch is invalid, and gives an error.

```
-- This batch is invalid.
BEGIN
  DECLARE local_var INT;
  SET local_var = 10;
END;
MESSAGE 'local_var = ', local_var TO CLIENT;
```

The following example illustrates the use of SELECT with an INTO clause to set the value of a local variable:

```
BEGIN
  DECLARE local_var INT;
```
SELECT 10 INTO local_var;
MESSAGE 'local_var = ', local_var TO CLIENT;
END

Running this batch from Interactive SQL displays the message local_var = 10 on the Messages tab of the Interactive SQL Results pane.

Standards and compatibility

- **SQL/2008** The DECLARE statement is supported in the ANSI SQL/2008 standard as SQL language feature P002, "Computational completeness".

See also

- "Column '%1' not found" [Error Messages]
- "Variables in Transact-SQL procedures" [SQL Anywhere Server - SQL Usage]

### Connection-level variables

Connection-level variables are declared with the CREATE VARIABLE statement. Connection-level variables can be passed as parameters to procedures.

The syntax for the CREATE VARIABLE statement is as follows:

```
CREATE VARIABLE variable-name data-type
```

When a variable is created, it is initially set to NULL. The value of connection-level variables can be set in the same way as local variables, using the SET statement or using a SELECT statement with an INTO clause.

Connection-level variables exist until the connection is terminated, or until the variable is explicitly dropped using the DROP VARIABLE statement. The following statement drops the variable con_var:

```
DROP VARIABLE con_var;
```

Example

The following batch of SQL statements illustrates the use of connection-level variables.

```
CREATE VARIABLE con_var INT;
SET con_var = 10;
MESSAGE 'con_var = ', con_var TO CLIENT;
```

Running this batch from Interactive SQL displays the message con_var = 10 on the Messages tab of the Interactive SQL Results pane.

Standards and compatibility

- **SQL/2008** Vendor extension.
Global variables

Global variables have values set by the database server. For example, the global variable @@version has a value that is the current version number of the database server.

Global variables are distinguished from local and connection-level variables by having two @ signs preceding their names. For example, @@error and @@rowcount are global variables. Users cannot create global variables, and cannot update the values of global variables directly.

Some global variables, such as @@identity, hold connection-specific information, and so have connection-specific values. Other variables, such as @@connections, have values that are common to all connections.

Global variable and special values

The special values (for example, CURRENT DATE, CURRENT TIME, USER, and SQLSTATE) are similar to global variables.

The following statement retrieves a value of the version global variable.

```
SELECT @@version;
```

In procedures and triggers, global variables can be selected into a variable list. The following procedure returns the server version number in the ver parameter.

```
CREATE PROCEDURE VersionProc ( OUT ver VARCHAR(100) )
BEGIN
  SELECT @@version
  INTO ver;
END;
```

In Embedded SQL, global variables can be selected into a host variable list.

Supported global variables

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@char_convert</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@client_csid</td>
<td>-1 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@client_csname</td>
<td>NULL (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@connections</td>
<td>The number of logins since the server was last started.</td>
</tr>
<tr>
<td>@@cpu_busy</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@dbts</td>
<td>A value of type TIMESTAMP representing the last generated value used for all columns defined with DEFAULT TIMESTAMP.</td>
</tr>
<tr>
<td>Variable name</td>
<td>Meaning</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>@@error</td>
<td>A Transact-SQL error code that checks the success or failure of the most recently executed statement. If the previous transaction succeeded, 0 is returned. If the previous transaction was unsuccessful, the last error number generated by the system is returned. For descriptions of the values returned by @@error, see “Error handling in Transact-SQL procedures” [SQL Anywhere Server - SQL Usage].</td>
</tr>
<tr>
<td></td>
<td>A statement such as if @@error != 0 return causes an exit if an error occurs. Every statement resets @@error, including PRINT statements or IF tests, so the status check must immediately follow the statement whose success you want verified.</td>
</tr>
<tr>
<td>@@fetch_status</td>
<td>Contains status information resulting from the last fetch statement. This feature is the same as @@sqlstatus, except that it returns different values. It is for Microsoft SQL Server compatibility. @@fetch_status may contain the following values:</td>
</tr>
<tr>
<td></td>
<td>● 0 The fetch statement completed successfully.</td>
</tr>
<tr>
<td></td>
<td>● -1 The fetch statement resulted in an error.</td>
</tr>
<tr>
<td></td>
<td>● -2 There is no more data in the result set.</td>
</tr>
<tr>
<td>@@identity</td>
<td>Last value inserted into any IDENTITY or DEFAULT AUTOINCREMENT column by an INSERT or SELECT INTO statement.</td>
</tr>
<tr>
<td>@@idle</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@io_busy</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@isolation</td>
<td>Current isolation level of the connection. @@isolation takes the value of the active level.</td>
</tr>
<tr>
<td>@@langid</td>
<td>Unique language ID for the language in use by the current connection.</td>
</tr>
<tr>
<td>@@language</td>
<td>Name of the language in use by the connection.</td>
</tr>
<tr>
<td>@@max_connections</td>
<td>For the personal server, the maximum number of simultaneous connections that can be made to the server, which is 10. For the network server, the maximum number of active clients (not database connections, as each client can support multiple connections).</td>
</tr>
<tr>
<td>@@maxcharlen</td>
<td>Maximum length, in bytes, of a character in the CHAR character set.</td>
</tr>
<tr>
<td>@@ncharsize</td>
<td>Maximum length, in bytes, of a character in the NCHAR character set.</td>
</tr>
<tr>
<td>@@nestlevel</td>
<td>-1 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>Variable name</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>@@pack_received</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@pack_sent</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@packet_errors</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@procid</td>
<td>Stored procedure ID of the currently executing procedure.</td>
</tr>
<tr>
<td>@@rowcount</td>
<td>Number of rows affected by the last statement. The value of @@rowcount should be checked immediately after the statement.</td>
</tr>
<tr>
<td></td>
<td>Inserts, updates, and deletes set @@rowcount to the number of rows affected.</td>
</tr>
<tr>
<td></td>
<td>With cursors, @@rowcount represents the cumulative number of rows returned from the cursor result set to the client, up to the last fetch request.</td>
</tr>
<tr>
<td></td>
<td>The @@rowcount is not reset to zero by any statement which does not affect rows, such as an IF statement.</td>
</tr>
<tr>
<td>@@servername</td>
<td>Name of the current database server.</td>
</tr>
<tr>
<td>@@spid</td>
<td>The connection handle for the current connection. This is the same value as that displayed by the sa_conn_info procedure.</td>
</tr>
<tr>
<td>@@sqlstatus</td>
<td>Contains status information resulting from the last fetch statement. @@sqlstatus may contain the following values:</td>
</tr>
<tr>
<td></td>
<td>● 0 The fetch statement completed successfully.</td>
</tr>
<tr>
<td></td>
<td>● 1 The fetch statement resulted in an error.</td>
</tr>
<tr>
<td></td>
<td>● 2 There is no more data in the result set.</td>
</tr>
<tr>
<td>@@textsize</td>
<td>Current value of the SET TEXTSIZE option, which specifies the maximum length, in bytes, of TEXT or IMAGE data to be returned with a SELECT statement. The default setting is 32765, which is the largest byte string that can be returned using READTEXT. The value can be set using the SET statement.</td>
</tr>
<tr>
<td>@@thresh_hysteresis</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@timeticks</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@total_errors</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
<tr>
<td>@@total_read</td>
<td>0 (Provided for compatibility with Transact-SQL.)</td>
</tr>
</tbody>
</table>
### @@total_write

0 (Provided for compatibility with Transact-SQL.)

### @@tranchained

Current transaction mode; 0 for unchained or 1 for chained.

### @@trancount

Nesting level of transactions. Each BEGIN TRANSACTION in a batch increments the transaction count.

### @@transtate

-1 (Provided for compatibility with Transact-SQL.)

### @@version

Version number of the current version of SQL Anywhere.

---

**See also**

- “@@identity global variable” on page 85

---

## @@identity global variable

The @@identity variable holds the most recent value inserted by the current connection into an IDENTITY column, a DEFAULT AUTOINCREMENT column, or a DEFAULT GLOBAL AUTOINCREMENT column, or zero if the most recent insert was into a table that had no such column.

The value of @@identity is connection specific. If a statement inserts multiple rows, @@identity reflects the IDENTITY value for the last row inserted. If the affected table does not contain an IDENTITY column, @@identity is set to zero.

The value of @@identity is not affected by the failure of an INSERT or SELECT INTO statement, or the rollback of the transaction that contained it. @@identity retains the last value inserted into an IDENTITY column, even if the statement that inserted it fails to commit.

### @@identity and triggers

When an insert causes referential integrity actions or fires a trigger, @@identity behaves like a stack. For example, if an insert into a table T1 (with an IDENTITY or AUTOINCREMENT column) fires a trigger that inserts a row into table T2 (also with an IDENTITY or AUTOINCREMENT column), then the value returned to the application or procedure which carried out the insert is the value inserted into T1. Within the trigger, @@identity has the T1 value before the insert into T2 and the T2 value after. The trigger can copy the values to local variables if it needs to access both.

### Standards and compatibility

- **SQL/2008**  Global variables are a vendor extension.

---

## Comments

Comments are used to attach explanatory text to SQL statements or statement blocks. The database server does not execute comments.
The following comment indicators are supported in SQL Anywhere:

- **-- (Double hyphen)**  The database server ignores any remaining characters on the line. This is the SQL/2008 comment indicator.

  You can add and remove this comment indicator by selecting text and pressing Ctrl+Minus Sign in Interactive SQL or on the SQL tab of the Procedures & Functions window of Sybase Central.

  The SQL comment indicator is added to the beginning of each line of the selected text. If no text is selected, the comment indicator is added to the beginning of the current line.

- **// (Double slash)**  The double slash has the same meaning as the double hyphen.

  You can add and remove this comment indicator by selecting text and pressing Ctrl+Forward Slash in Interactive SQL or on the SQL tab of the Procedures & Functions window of Sybase Central.

  The SQL comment indicator is added to the beginning of each line of the selected text. If no text is selected, the comment indicator is added to the beginning of the current line.

- **/* ... */ (Slash-asterisk)**  Any characters between the two comment markers are ignored. The two comment markers can be on the same or different lines. Comments indicated in this style can be nested, but nested comments must be balanced. Any comments made inside the comment block must not contain single instances of the starting comment marker. This style of commenting is also called C-style comments.

  Examples

  The following example illustrates the use of double-hyphen comments:

  ```sql
  CREATE FUNCTION fullname ( firstname CHAR(30),
      lastname CHAR(30))
  RETURNS CHAR(61)
  -- fullname concatenates the firstname and lastname
  -- arguments with a single space between.
  BEGIN
      DECLARE name CHAR(61);
      SET name = firstname || ' ' || lastname;
      RETURN ( name );
  END;
  ```

  The following example illustrates the use of C-style comments:

  ```sql
  /* Lists the names and employee IDs of employees
  who work in the sales department. */
  CREATE VIEW SalesEmployees AS
      SELECT EmployeeID, Surname, GivenName
      FROM GROUPO.Employees
      WHERE DepartmentID = 200;
  ```

Standards and compatibility

- **SQL/2008**  The use of double-minus signs for a comment is a core feature of the ANSI SQL/2008 standard. The use of C-style, bracketed comments (/* ... */f) is SQL language feature T351 of the SQL/2008 standard. Double-slash comments (//) are supported as a vendor extension.
Named parameters

SQL Anywhere functions and procedures that are referenced from the CALL statement, the EXECUTE statement (Transact-SQL), or the FROM clause of a DML statement support positional parameters and named parameters. Named parameters allow you to specify any subset of the available parameters in any order. You can use named parameters with system-defined procedures and functions and with user-defined procedures. Named parameters cannot be used with user-defined functions.

The following named parameter syntaxes are supported:

- =
- =>

Example

The following example uses = to specify a named parameter:

```sql
CALL sa_conn_properties( connidparm = 1 );
```

This example uses named parameters when calling the system procedure, and returns information about the remote tables with foreign keys on the SYSOBJECTS table, in the production database, on a database server named satest:

```sql
CALL sp_remote_exported_keys(
    @server_name=>'satest',
    @sp_name=>'sysobjects',
    @sp_qualifier=>'production' );
```

Standards and compatibility

- SQL/2008  = and => are vendor extensions.

See also

- “CALL statement” on page 530
- “EXECUTE IMMEDIATE statement [SP]” on page 791
- “EXECUTE statement [ESQL]” on page 794
- “EXECUTE statement [T-SQL]” on page 796
- “FROM clause” on page 810
- “The CLR external environment” [SQL Anywhere Server - Programming]
SQL data types

Character data types

Character data types store strings of letters, numbers, and other symbols.

SQL Anywhere provides two classes of character data types and some domains defined using those types.

- **CHAR, VARCHAR, LONG VARCHAR** Character data stored in a single- or multibyte character set, often chosen to correspond most closely to the primary language or languages stored in the database.

- **NCHAR, NVARCHAR, LONG NVARCHAR** Character data stored in the UTF-8 Unicode encoding. All Unicode code points can be stored using these types, regardless of the primary language or languages stored in the database.

- **TEXT, UNIQUEIDENTIFIERSTR, XML** Domains based on other character data types.

Storage

All character data values are stored in the same manner. By default, values up to 128 bytes are stored in a single piece. Values longer than 128 bytes are stored with a 4-byte prefix kept locally on the database page and the full value stored in one or more other database pages. These default sizes are controlled by the INLINE and PREFIX clauses of the CREATE TABLE statement.

See also

- “CREATE TABLE statement” on page 690
- “string_rtruncation option” [SQL Anywhere Server - Database Administration]

CHAR data type

The CHAR data type stores character data, up to 32767 bytes.

Syntax

```sql
CHAR [ ( max-length [ CHAR | CHARACTER ] ) ]
```

Parameters

- **max-length** The maximum length of the string. If byte-length semantics are used (CHAR or CHARACTER is not specified as part of the length), then the length is in bytes, and the length must be in the range 1 to 32767. If the length is not specified, then it is 1.

If character-length semantics are used (CHAR or CHARACTER is specified as part of the length), then the length is in characters, and you must specify `max-length`. `max-length` can be a maximum of 32767 characters.
Remarks

Multibyte characters can be stored as CHAR, but the declared length refers to bytes, not characters, unless character-length semantics are used.

CHAR can also be specified as CHARACTER. Regardless of which syntax is used, the data type is described as CHAR.

CHAR is semantically equivalent to VARCHAR, although they are different types. In SQL Anywhere, CHAR is a variable-length type. In other relational database management systems, CHAR is a fixed-length type, and data is padded with blanks to max-length bytes of storage. SQL Anywhere does not blank-pad stored character data.

How CHAR columns are described depends on the client interface, the character sets used, and if character-length semantics are used. For example, in embedded SQL the described length is the maximum number of bytes in the client character set. If the described length would be more than 32767 bytes, the column is described as type DT_LONGVARCHAR. The following table shows some embedded SQL examples and the results returned when a DESCRIBE is performed:

<table>
<thead>
<tr>
<th>Type being described</th>
<th>Database character set</th>
<th>Client character set</th>
<th>Result of DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(10)</td>
<td>Windows-1252</td>
<td>Windows-1252</td>
<td>DT_FIXCHAR length 10</td>
</tr>
<tr>
<td>CHAR(10)</td>
<td>UTF-8</td>
<td>UTF-8</td>
<td>DT_FIXCHAR length 10</td>
</tr>
<tr>
<td>CHAR(10)</td>
<td>Windows-1252</td>
<td>UTF-8</td>
<td>DT_FIXCHAR length 30</td>
</tr>
<tr>
<td>CHAR(20000)</td>
<td>Windows-31J</td>
<td>UTF-8</td>
<td>DT_LONGVARCHAR</td>
</tr>
<tr>
<td>CHAR(10 CHAR )</td>
<td>Windows-1252</td>
<td>Windows-1252</td>
<td>DT_FIXCHAR length 10</td>
</tr>
<tr>
<td>CHAR(10 CHAR )</td>
<td>UTF-8</td>
<td>UTF-8</td>
<td>DT_FIXCHAR length 40</td>
</tr>
</tbody>
</table>

For ODBC, CHAR is described as either SQL_CHAR or SQL_VARCHAR depending on the odbc_distinguish_char_and_varchar option.

See also

- “odbc_distinguish_char_and_varchar option” [SQL Anywhere Server - Database Administration]
- “VARCHAR data type” on page 95
- “LONG VARCHAR data type” on page 91
- “NCHAR data type” on page 92

Standards and compatibility

- **SQL/2008** Compatible with SQL/2008. In the standard, character-length semantics are the default, whereas in SQL Anywhere byte-length semantics are the default. There are minor inconsistencies with the SQL standard due to case-insensitive collation support and SQL Anywhere blank-padding support.

  The SQL/2008 standard supports explicit character- or byte-length semantics as SQL language feature T061.
LONG NVARCHAR data type

The LONG NVARCHAR data type stores Unicode character data of arbitrary length.

Syntax

LONG NVARCHAR

Remarks

The maximum size is 2 GB minus 1 byte \(2^{31} - 1\).

Characters are stored in UTF-8. Each character requires from one to four bytes. The maximum number of characters that can be stored in a LONG NVARCHAR is over 500 million and possibly over 2 billion, depending on the lengths of the characters stored.

When an embedded SQL client performs a DESCRIBE on a LONG NVARCHAR column, the data type returned is either DT_LONGVARCHAR or DT_LONGNVARCHAR, depending on whether the db_change_nchar_charset function has been called.

For ODBC, a LONG NVARCHAR expression is described as SQL_WLONGVARCHAR.

See also

- “db_change_nchar_charset function” [SQL Anywhere Server - Programming]
- “NCHAR data type” on page 92
- “NVARCHAR data type” on page 93
- “LONG VARCHAR data type” on page 91

Standards and compatibility

- SQL/2008  The use of LONG NVARCHAR to declare a national character string is a vendor extension.

LONG VARCHAR data type

The LONG VARCHAR data type stores character data of arbitrary length.

Syntax

LONG VARCHAR

Remarks

The maximum byte size is 2 GB minus 1 byte \(2^{31} - 1\).

Multibyte characters can be stored as LONG VARCHAR, but the length is in bytes, not characters.

See also

- “CHAR data type” on page 89
- “VARCHAR data type” on page 95
- “LONG NVARCHAR data type” on page 91
Standards and compatibility

- SQL/2008  Large object support is SQL language feature T041 of the SQL/2008 standard.

NCHAR data type

The NCHAR data type stores Unicode character data, up to 32767 characters.

Syntax

```
NCHAR [ ( max-length ) ]
```

Parameters

- **max-length**  The maximum length of the string, in characters. The length must be in the range 1 to 32767. If the length is not specified, then it is 1.

Remarks

Characters are stored using UTF-8 encoding. The maximum number of bytes of storage required is four multiplied by `max-length`. However, the actual number of bytes of storage required is usually much less.

NCHAR can also be specified as NATIONAL CHAR or NATIONAL CHARACTER. Regardless of which syntax is used, the data type is described as NCHAR.

When an embedded SQL client performs a DESCRIBE on an NCHAR column, the data type returned is either DT_FIXCHAR or DT_NFIXCHAR, depending on whether the `db_change_nchar_charset` function has been called.

Also, when an embedded SQL client performs a DESCRIBE on an NCHAR column, the length returned is the maximum byte length in the client NCHAR character set. For example, for an embedded SQL client using the Western European character set cp1252 as the NCHAR character set, an NCHAR(10) column is described as type DT_NFIXCHAR of length 10 (10 characters multiplied by a maximum one byte per character). For an embedded SQL client using the Japanese character set cp932, the same column is described as type DT_NFIXCHAR of length 20 (10 characters multiplied by a maximum two bytes per character). If the described length would return more than 32767 bytes, the column is described as type DT_LONGNVARCHAR.

NCHAR is semantically equivalent to NVARCHAR, although they are different types. In SQL Anywhere, NCHAR is a variable-length type. In other relational database management systems, NCHAR is a fixed-length type, and data is padded with blanks to `max-length` characters of storage. SQL Anywhere does not blank-pad stored character data.

For ODBC, NCHAR is described as SQL_WCHAR.

See also

- “`db_change_nchar_charset` function” [SQL Anywhere Server - Programming]
- “CHAR data type” on page 89
- “NVARCHAR data type” on page 93
- “LONG NVARCHAR data type” on page 91
Standards and compatibility

- SQL/2008  National character support is feature F421 of the SQL/2008 standard.

**NTEXT data type**

The NTEXT data type stores Unicode character data of arbitrary length.

**Syntax**

NTEXT

**Remarks**

NTEXT is a domain, implemented as a LONG NVARCHAR.

**See also**

- “LONG NVARCHAR data type” on page 91
- “TEXT data type” on page 94

Standards and compatibility

- SQL/2008  Vendor extension.

**NVARCHAR data type**

The NVARCHAR data type stores Unicode character data, up to 32767 characters.

**Syntax**

NVARCHAR [ ( max-length ) ]

**Parameters**

- **max-length**  The maximum length of the string, in characters. The length must be in the range 1 to 32767. If the length is not specified, then it is 1.

**Remarks**

Characters are stored in UTF-8 encoding. The maximum storage number of bytes required is four multiplied by max-length, although the actual storage required is usually much less.

NVARCHAR can also be specified as NCHAR VARYING, NATIONAL CHAR VARYING, or NATIONAL CHARACTER VARYING. Regardless of which syntax is used, the data type is described as NVARCHAR.

When an embedded SQL client performs a DESCRIBE on a NVARCHAR column, the data type returned is either DT_VARCHAR or DT_NVARCHAR, depending on whether the db_change_nchar_charset function has been called.

Also, when an embedded SQL client performs a DESCRIBE on an NVARCHAR column, the length returned is the maximum byte length in the client NCHAR character set. For example, for an embedded...
SQL client using the Western European character set cp1252 as the NCHAR character set, an
NVARCHAR(10) column is described as type DT_NVARCHAR of length 10 (10 characters multiplied
by a maximum of one byte per character). For an embedded SQL client using the Japanese character set
cp932, the same column is described as type DT_NVARCHAR of length 20 (10 characters multiplied by
a maximum two bytes per character). If the describe length would return more than 32767 bytes, the
column is described as type DT_LONGNVARCHAR.

For ODBC, NVARCHAR is described as SQL_WVARCHAR.

See also
● “db_change_nchar_charset function” [SQL Anywhere Server - Programming]
● “NCHAR data type” on page 92
● “LONG NVARCHAR data type” on page 91
● “VARCHAR data type” on page 95

Standards and compatibility
● SQL/2008 National character support is SQL language feature F421 in the SQL/2008 standard.

TEXT data type

The TEXT data type stores character data of arbitrary length.

Syntax
TEXT

Remarks
TEXT is a domain, implemented as a LONG VARCHAR.

See also
● “LONG VARCHAR data type” on page 91
● “NTEXT data type” on page 93

Standards and compatibility
● SQL/2008 Vendor extension.

UNIQUEIDENTIFIERSTR data type

UNIQUEIDENTIFIERSTR is a domain, implemented as CHAR(36).

Syntax
UNIQUEIDENTIFIERSTR

Remarks
Used for remote data access, when mapping Microsoft SQL Server uniqueidentifier columns.
VARCHAR data type

The VARCHAR data type stores character data, up to 32767 bytes.

Syntax

```
VARCHAR [( max-length [ CHAR | CHARACTER ] )]
```

Parameters

- **max-length**  The maximum length of the string. This default value is 1.

  If byte-length semantics are used (CHAR or CHARACTER is *not* specified as part of the length), then the length is in bytes, and the length must be in the range of 1 to 32767.

  If character-length semantics are used (CHAR or CHARACTER is specified as part of the length), then the length is in characters, and you must specify `max-length`. `max-length` can be a maximum of 32767 characters.

Remarks

Multibyte characters can be stored as VARCHAR, but the declared length refers to bytes, not characters.

VARCHAR can also be specified as CHAR VARYING or CHARACTER VARYING. Regardless of which syntax is used, the data type is described as VARCHAR.

VARCHAR is semantically equivalent to CHAR, although they are different types. In SQL Anywhere, VARCHAR is a variable-length type. In other relational database management systems, VARCHAR is a fixed-length type, and data is padded with blanks to `max-length` bytes of storage. SQL Anywhere does not blank-pad stored character data.

How VARCHAR columns are described depends on the client interface, the character sets used, and if character-length semantics are used. For example, in embedded SQL the described length is the maximum number of bytes in the client character set. If the described length would be more than 32767 bytes, the column is described as type DT_LONGVARCHAR. The following table shows some embedded SQL examples and the results returned when a DESCRIBE is performed:

<table>
<thead>
<tr>
<th>Type being described</th>
<th>Database character set</th>
<th>Client character set</th>
<th>Result of DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR(10)</td>
<td>Windows-1252</td>
<td>Windows-1252</td>
<td>DT_VARCHAR length 10</td>
</tr>
</tbody>
</table>
### SQL data types

<table>
<thead>
<tr>
<th>Type being described</th>
<th>Database character set</th>
<th>Client character set</th>
<th>Result of DESCRIBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR(10)</td>
<td>UTF-8</td>
<td>UTF-8</td>
<td>DT_VARCHAR length 10</td>
</tr>
<tr>
<td>VARCHAR(10)</td>
<td>Windows-1252</td>
<td>UTF-8</td>
<td>DT_VARCHAR length 30</td>
</tr>
<tr>
<td>VARCHAR(20000)</td>
<td>Windows-31J</td>
<td>UTF-8</td>
<td>DT_LONGVARCHAR</td>
</tr>
<tr>
<td>VARCHAR(10 CHAR )</td>
<td>Windows-1252</td>
<td>Windows-1252</td>
<td>DT_VARCHAR length 10</td>
</tr>
<tr>
<td>VARCHAR(10 CHAR )</td>
<td>UTF-8</td>
<td>UTF-8</td>
<td>DT_VARCHAR length 40</td>
</tr>
</tbody>
</table>

For ODBC, VARCHAR is described as SQL_VARCHAR.

### See also
- “CHAR data type” on page 89
- “LONG VARCHAR data type” on page 91
- “NVARCHAR data type” on page 93

### Standards and compatibility

- **SQL/2008** Compatible with SQL/2008. In the standard, character-length semantics are the default, whereas in SQL Anywhere byte-length semantics are the default. There are minor inconsistencies with the SQL standard due to case-insensitive collation support and SQL Anywhere blank-padding support.

  The SQL/2008 standard supports explicit character- or byte-length semantics as SQL language feature T061.

### XML data type

The XML data type stores character data of arbitrary length, and stores XML documents.

**Syntax**

`XML`

**Remarks**

The maximum size is 2 GB minus 1 byte ($2^{31} - 1$).

Data of type XML is not quoted when generating element content from relational data.

You can cast between the XML data type and any other data type that can be cast to or from a string. There is no checking that the string is well-formed when it is cast to XML.

When an embedded SQL client application performs a DESCRIBE on an XML column, it is described as LONG VARCHAR.
See also

- “XML in the database” [SQL Anywhere Server - SQL Usage]
- “Storage of XML documents in relational databases” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- **SQL/2008**  The XML data type is SQL language feature X010 in the SQL/2008 standard.

## Numeric data types

Numeric data types store numerical data.

The NUMERIC and DECIMAL data types, and the various INTEGER data types, are sometimes called **exact** numeric data types, in contrast to the **approximate** numeric data types FLOAT, DOUBLE, and REAL.

The exact numeric data types are those for which precision and scale values can be specified, while approximate numeric data types are stored in a predefined manner. *Only exact numeric data is guaranteed accurate to the least significant digit specified after an arithmetic operation.*

Data type lengths and precision of less than one are not allowed.

**Compatibility**

Only the NUMERIC data type with scale = 0 can be used for the Transact-SQL identity column.

Be careful using default precision and scale settings for NUMERIC and DECIMAL data types because these settings could be different in other database solutions. The default precision is 30 and the default scale is 6.

You should avoid default precision and scale settings for NUMERIC and DECIMAL data types, because these are different between SQL Anywhere and Adaptive Server Enterprise. In SQL Anywhere, the default precision is 30 and the default scale is 6. In Adaptive Server Enterprise, the default precision is 18 and the default scale is 0.

The FLOAT (p) data type is a synonym for REAL or DOUBLE, depending on the value of p. For SQL Anywhere, the cutoff is platform-dependent, but on all platforms the cutoff value is greater than 15.

For information about changing the defaults by setting database options, see “precision option” [SQL Anywhere Server - Database Administration] and “scale option” [SQL Anywhere Server - Database Administration].

## BIGINT data type

The BIGINT data type stores BIGINTs, which are integers requiring 8 bytes of storage.

**Syntax**

```
[ UNSIGNED ] BIGINT
```
Remarks

The BIGINT data type is an exact numeric data type: its accuracy is preserved after arithmetic operations.

A BIGINT value requires 8 bytes of storage.

The range for BIGINT values is \(-2^{63}\) to \(2^{63} - 1\), or \(-9223372036854775808\) to \(9223372036854775807\).

The range for UNSIGNED BIGINT values is \(0\) to \(2^{64} - 1\), or \(0\) to \(18446744073709551615\).

By default, the data type is signed.

When converting a string to a BIGINT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

See also

- “BIT data type” on page 98
- “INTEGER data type” on page 102
- “SMALLINT data type” on page 105
- “TINYINT data type” on page 105
- “Numeric functions” on page 151
- “Aggregate functions” on page 143

Standards and compatibility

- SQL/2008  The BIGINT data type is SQL language feature T071 of the SQL/2008 standard.
- MySQL  The UNSIGNED keyword may follow BIGINT.

BIT data type

The BIT data type stores a bit (0 or 1).

Syntax

BIT

Remarks

BIT is an integer type that can store the values 0 or 1.

By default, the BIT data type does not allow NULL.

A BIT value requires 1 byte of storage.

When converting a string to a BIT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is not 0 or 1.
See also

- “BIGINT data type” on page 97
- “INTEGER data type” on page 102
- “SMALLINT data type” on page 105
- “TINYINT data type” on page 105
- “Numeric functions” on page 151
- “Aggregate functions” on page 143

Standards and compatibility

- SQL/1999 The BIT data type is SQL language feature F511 of the SQL/1999 standard.

- SQL/2008 The BIT and BIT VARYING data types were dropped from the SQL/2003 standard. Hence with respect to the SQL/2008 standard, the BIT data type is a vendor extension.

DECIMAL data type

The DECIMAL data type is a decimal number with precision total digits and with scale digits after the decimal point.

Syntax

```sql
DECIMAL [(precision [ , scale ]) ]
```

Parameters

- **precision** An integer expression between 1 and 127, inclusive, that specifies the number of digits in the expression. The default setting is 30.

- **scale** An integer expression between 0 and 127, inclusive, that specifies the number of digits after the decimal point. The scale value should always be less than, or equal to, the precision value. The default setting is 6.

  The defaults can be changed by setting database options.

Remarks

The DECIMAL data type is an exact numeric data type; its accuracy is preserved to the least significant digit after arithmetic operations.

The number of bytes required to store a decimal number can be estimated as

\[
2 + \text{INT}((\text{precision} - \text{scale}) + 1) / 2) + \text{INT}((\text{scale} + 1) / 2);
\]

The INT function takes the integer portion of its argument. The storage is based on the value being stored, not on the maximum precision and scale allowed in the column.

DECIMAL can also be specified as DEC. Regardless of which syntax is used, the data type is described as DECIMAL.

If you are using a precision of 20 or less and a scale of 0, it may be possible to use one of the integer data types (BIGINT, INTEGER, SMALLINT, or TINYINT) instead. Integer values require less storage space
than NUMERIC and DECIMAL values with a similar number of significant digits. Operations on integer values, such as fetching or inserting, and arithmetic operators, typically perform better than operations on NUMERIC and DECIMAL values.

DECIMAL is semantically equivalent to NUMERIC.

Note
If you create a column or variable of a DECIMAL data type with a precision or scale that exceeds the precision and scale settings for the database, values are truncated to the database settings. So, if you notice truncated values in a column or variable defined as DECIMAL, check that precision and scale do not exceed the database option settings.

See also
● “FLOAT data type” on page 101
● “REAL data type” on page 104
● “DOUBLE data type” on page 100
● “NUMERIC data type” on page 103
● “Numeric functions” on page 151
● “Aggregate functions” on page 143
● “precision option” [SQL Anywhere Server - Database Administration]
● “scale option” [SQL Anywhere Server - Database Administration]

Standards and compatibility
● SQL/2008 DECIMAL and NUMERIC data types are core features of the SQL/2008 standard.

DOUBLE data type
The DOUBLE data type stores double-precision floating-point numbers.

Syntax
DOUBLE

Remarks
The DOUBLE data type is an approximate numeric data type and subject to rounding errors after arithmetic operations. The approximate nature of DOUBLE values means that queries using equalities should generally be avoided when comparing DOUBLE values.

DOUBLE values require 8 bytes of storage.

The range of values is -1.79769313486231e+308 to 1.79769313486231e+308, with numbers close to zero as small as 2.22507385850721e-308. Values held as DOUBLE are accurate to 15 significant digits, but may be subject to rounding errors beyond the fifteenth digit.
See also

- “FLOAT data type” on page 101
- “REAL data type” on page 104
- “DECIMAL data type” on page 99
- “NUMERIC data type” on page 103
- “Numeric functions” on page 151
- “Numeric set conversions” on page 140
- “Aggregate functions” on page 143

Standards and compatibility

- SQL/2008 The DOUBLE PRECISION type is a core feature of the SQL/2008 standard.

FLOAT data type

The FLOAT data type stores a floating-point number, which can be single or double precision.

Syntax

FLOAT [ ( precision ) ]

Parameters

- precision An integer expression that specifies the number of bits in the mantissa, the decimal part of a logarithm. For example, in the number 5.63428, the mantissa is 0.63428. The IEEE standard 754 floating-point precision is as follows:

<table>
<thead>
<tr>
<th>Supplied precision value</th>
<th>Decimal precision</th>
<th>Equivalent SQL data type</th>
<th>Storage size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-24</td>
<td>7 decimal digits</td>
<td>REAL</td>
<td>4 bytes</td>
</tr>
<tr>
<td>25-53</td>
<td>15 decimal digits</td>
<td>DOUBLE</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

Remarks

When a column is created using the FLOAT ( precision ) data type, columns on all platforms are guaranteed to hold the values to at least the specified minimum precision. REAL and DOUBLE do not guarantee a platform-independent minimum precision.

If precision is not supplied, the FLOAT data type is a single-precision floating-point number, equivalent to the REAL data type, and requires 4 bytes of storage.

If precision is supplied, the FLOAT data type is either single or double precision, depending on the value of precision specified. The cutoff between REAL and DOUBLE is platform-dependent. Single-precision FLOAT values require 4 bytes of storage, and double-precision FLOAT values require 8 bytes.

The FLOAT data type is an approximate numeric data type. It is subject to rounding errors after arithmetic operations. The approximate nature of FLOAT values means that queries using equalities should be avoided when comparing FLOAT values.
See also

- “DOUBLE data type” on page 100
- “REAL data type” on page 104
- “DECIMAL data type” on page 99
- “NUMERIC data type” on page 103
- “Numeric functions” on page 151
- “Aggregate functions” on page 143

Standards and compatibility

- SQL/2008 The FLOAT type is a core feature of the SQL/2008 standard.

**INTEGER data type**

The INTEGER data type stores integers that require 4 bytes of storage.

**Syntax**

```sql
[UNSIGNED] INTEGER
```

**Remarks**

The INTEGER data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

If you specify UNSIGNED, the integer can never be assigned a negative number. By default, the data type is signed.

The range for INTEGER values is \(-2^{31} \) to \(2^{31} - 1\), or \(-2147483648 \) to \(2147483647\).

The range for UNSIGNED INTEGER values is \(0 \) to \(2^{32} - 1\), or \(0 \) to \(4294967295\).

When converting a string to an INTEGER, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

See also

- “BIGINT data type” on page 97
- “BIT data type” on page 98
- “SMALLINT data type” on page 105
- “TINYINT data type” on page 105
- “Numeric functions” on page 151
- “Aggregate functions” on page 143

Standards and compatibility

- SQL/2008 The INTEGER type is a core feature of the SQL/2008 standard. The UNSIGNED keyword is a vendor extension.
NUMERIC data type

The NUMERIC data type stores decimal numbers with precision total digits and with scale digits after the decimal point.

Syntax

```
NUMERIC [ ( precision [ , scale ] ) ]
```

Parameters

- **precision** An integer expression between 1 and 127, inclusive, that specifies the number of digits in the expression. The default setting is 30.

- **scale** An integer expression between 0 and 127, inclusive, that specifies the number of digits after the decimal point. The scale value should always be less than or equal to the precision value. The default setting is 6.

Remarks

The NUMERIC data type is an exact numeric data type; its accuracy is preserved to the least significant digit after arithmetic operations.

The number of bytes required to store a decimal number can be estimated as

```
2 + INT( (BEFORE+1)/2 ) + INT( (AFTER+1)/2 );
```

The INT function takes the integer portion of its argument, and BEFORE and AFTER are the number of significant digits before and after the decimal point. The storage is based on the value being stored, not on the maximum precision and scale allowed in the column.

If you are using a precision of 20 or less and a scale of 0, it may be possible to use one of the integer data types (BIGINT, INTEGER, SMALLINT, or TINYINT) instead. Integer values require less storage space than NUMERIC and DECIMAL values with a similar number of significant digits. Operations on integer values, such as fetching or inserting, and arithmetic operators, typically perform better than operations on NUMERIC and DECIMAL values.

NUMERIC is semantically equivalent to DECIMAL.

**Note**

If you create a column or variable of a NUMERIC data type with a precision or scale that exceeds the precision and scale settings for the database, values are truncated to the database settings. So, if you notice truncated values in a column or variable defined as NUMERIC, check that precision and scale do not exceed the database option settings.
See also

- “FLOAT data type” on page 101
- “REAL data type” on page 104
- “DOUBLE data type” on page 100
- “DECIMAL data type” on page 99
- “Numeric functions” on page 151
- “Aggregate functions” on page 143
- “Numeric set conversions” on page 140
- “precision option” [SQL Anywhere Server - Database Administration]
- “scale option” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Compatible with SQL/2008 if the scale option is set to zero.

REAL data type

The REAL data type stores single-precision floating-point numbers stored in 4 bytes.

Syntax

REAL

Remarks

The REAL data type is an approximate numeric data type and subject to rounding errors after arithmetic operations. The approximate nature of REAL values means that queries using equalities should generally be avoided when comparing REAL values.

REAL values require 4 bytes of storage.

The range of values is -3.402823e+38 to 3.402823e+38, with numbers close to zero as small as 1.175494351e-38. Values held as REAL are accurate to 7 significant digits, but may be subject to rounding error beyond the sixth digit.

See also

- “DOUBLE data type” on page 100
- “FLOAT data type” on page 101
- “DECIMAL data type” on page 99
- “NUMERIC data type” on page 103
- “Numeric functions” on page 151
- “Aggregate functions” on page 143

Standards and compatibility

- SQL/2008 The REAL data type is a core feature of the SQL/2008 standard.
**SMALLINT data type**

The SMALLINT data type stores integers that require 2 bytes of storage.

**Syntax**

```
[ UNSIGNED ] SMALLINT
```

**Remarks**

The SMALLINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations. It requires 2 bytes of storage.

The range for SMALLINT values is $-2^{15}$ to $2^{15} - 1$, or -32768 to 32767.

The range for UNSIGNED SMALLINT values is 0 to $2^{16} - 1$, or 0 to 65535.

When converting a string to a SMALLINT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

**See also**

- “BIGINT data type” on page 97
- “BIT data type” on page 98
- “INTEGER data type” on page 102
- “TINYINT data type” on page 105
- “Numeric functions” on page 151
- “Aggregate functions” on page 143

**Standards and compatibility**

- **SQL/2008**  Compatible with SQL/2008. The UNSIGNED keyword is a vendor extension.

- **MySQL**  The UNSIGNED keyword may follow SMALLINT.

**TINYINT data type**

The TINYINT data type stores unsigned integers requiring 1 byte of storage.

**Syntax**

```
TINYINT
```

**Remarks**

The TINYINT data type is an exact numeric data type; its accuracy is preserved after arithmetic operations.

The range for TINYINT values is 0 to $2^8 - 1$, or 0 to 255.
In embedded SQL, TINYINT columns should not be fetched into variables defined as CHAR or UNSIGNED CHAR, since the result is an attempt to convert the value of the column to a string and then assign the first byte to the variable in the program. Instead, TINYINT columns should be fetched into 2-byte or 4-byte integer columns. To send a TINYINT value to a database from an application written in C, the type of the C variable should be INTEGER.

When converting a string to a TINYINT, leading and trailing spaces are removed. If the leading character is +, it is ignored. If the leading character is -, the remaining digits are interpreted as a negative number. Leading 0 characters are skipped, and the remaining characters are converted to an integer value. An error is returned if the value is out of the valid range for the destination data type, if the string contains illegal characters, or if the string cannot be decoded as an integer value.

See also

- “BIGINT data type” on page 97
- “BIT data type” on page 98
- “INTEGER data type” on page 102
- “SMALLINT data type” on page 105
- “Numeric functions” on page 151
- “Aggregate functions” on page 143

Standards and compatibility

- SQL/2008   Vendor extension.
- MySQL   The UNSIGNED keyword may precede or follow TINYINT, but the UNSIGNED modifier has no effect as the type is always unsigned.

Money data types

Money data types are used for storing monetary data.

MONEY data type

The MONEY data type stores monetary data.

Syntax

MONEY

Remarks

MONEY is a domain, implemented as NUMERIC(19,4).

See also

- “SMALLMONEY data type” on page 107
- “Numeric functions” on page 151
- “Aggregate functions” on page 143
Standards and compatibility
- SQL/2008  Vendor extension.

SMALLMONEY data type
The SMALLMONEY data type stores monetary data that is less than one million currency units.

Syntax
SMALLMONEY

Remarks
SMALLMONEY is a domain, implemented as NUMERIC(10,4).

See also
- “MONEY data type” on page 106
- “Numeric functions” on page 151
- “Aggregate functions” on page 143

Bit array data types
A bit array is similar to a character string, except that the individual pieces are bit data—0s (zeros) and 1s
(ones) instead of characters. Typically, bit arrays are used to hold a string of Boolean values.

The bit array data types supported by SQL Anywhere include VARBIT and LONG VARBIT.

LONG VARBIT data type
The LONG VARBIT data type stores arbitrary length bit arrays.

Syntax
LONG VARBIT

Remarks
Used to store arbitrary length array of bits (1s and 0s), or bit arrays longer than 32767 bits.

LONG VARBIT can also be specified as LONG BIT VARYING. Regardless of which syntax is used, the
data type is described as LONG VARBIT.
VARBIT data type

The VARBIT data type is used for storing bit arrays that are under 32767 bits in length.

Syntax

VARBIT [ (max-length) ]

Parameters

- **max-length**  The maximum length of the bit array, in bits. The length must be in the range 1 to 32767. If the length is not specified, then it is 1.

Remarks

VARBIT can also be specified as BIT VARYING. Regardless of which syntax is used, the data type is described as VARBIT.

See also

- “BIT data type” on page 98
- “VARBIT data type” on page 108
- “Bit array conversions” on page 139
- “Bit array functions” on page 145
- “Aggregate functions” on page 143

Standards and compatibility

- **SQL/2008**  Vendor extension.

Date and time data types

The following list provides a quick overview of how dates are handled:
Correct values are always returned for any legal arithmetic and logical operations on dates, regardless of whether the calculated values span different centuries.

The internal storage of dates always explicitly includes the century portion of a year value.

Date values can always be output in full century format.

**How date and time of day are stored**

Date and time of day are stored in SQL Anywhere databases using one of the following data types:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Contains</th>
<th>Stored in</th>
<th>Range of possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>Calendar date (year, month, day)</td>
<td>4 bytes</td>
<td>Dates from 0001-01-01 to 9999-12-31.</td>
</tr>
<tr>
<td>TIME</td>
<td>Time of day (hour, minute, second, and fraction of a second accurate to 6 decimal places)</td>
<td>8 bytes</td>
<td>Times from 00:00:00:00000000 to 24:00:00:000000.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Calendar date and time of day (year, month, day, hour, minute, second, and fraction of a second accurate to 6 decimal places)</td>
<td>8 bytes</td>
<td>Dates from 0001-01-01 to 9999-12-31 (precision of the hours and minutes portion of a TIMESTAMP is dropped before 1600-02-28 23:59:59 and after 7911-01-01 00:00:00).</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Calendar date, time of day, and time zone offset (year, month, day, hour, minute, second, fraction of a second accurate to 6 decimal places, and time zone offset in hours and minutes)</td>
<td>10 bytes</td>
<td>Dates from 0001-01-01 to 9999-12-31 (precision of the hours and minutes portion of a TIMESTAMP WITH TIME ZONE is dropped before 1600-02-28 23:59:59 and after 7911-01-01 00:00:00). Zone offset from -14:59 to +14:59.</td>
</tr>
</tbody>
</table>

**Ways to send dates and times to the database**

The date and time of day can be sent to the database in one of the following ways:

- using any interface, as a string
- using ODBC or OLE DB, as a binary value (using an ODBC TIMESTAMP_STRUCT structure for example)
- using embedded SQL, as a SQLDATETIME structure

The date and time of day with a time zone offset can be sent to the database as a string only.
Date formats

When a date is sent to the database as a string (for the DATE data type) or as part of a string (for the
TIMESTAMP or TIMESTAMP WITH TIME ZONE data types), the string can be specified in a number
of different ways including that described by ISO 8601, an international standard on the representation of
dates and times.

A date can be specified in one of the following ISO 8601 formats.

- **Calendar date**  The calendar date format is YYYY-MM-DD where YYYY is the year in the
  Gregorian calendar, MM is the month of the year between 01 (January) and 12 (December), and DD is
  the day of the month between 01 and 31. For example, '2010-04-01' represents the first day of April in
  2010. ISO 8601 does not require the separator character. Therefore, '20100401' also represents the
  first day of April in 2010.

<table>
<thead>
<tr>
<th>ISO calendar date</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>YYYYMMDD</td>
<td>20100401</td>
</tr>
<tr>
<td>Extended</td>
<td>YYYY-MM-DD</td>
<td>2010-04-01</td>
</tr>
</tbody>
</table>

- **Week date**  Another ISO date format is the week date. The format is YYYY-Www-D where
  YYYY is the year in the Gregorian calendar, W is the letter W, ww is the week of the year between
  01 (the first week) and 52 or 53 (the last week), and D is the day in the week between 1 (Monday) and
  7 (Sunday). For example, '2010-W13-4' represents the fourth day of the thirteenth week of 2010 (April
  1 2010). ISO 8601 does not require the separator character. Therefore, '2010W134' also represents the
  fourth day of the thirteenth week of 2010. For reduced accuracy, one digit can be omitted from the
  representation ('2010W13' represents March 29 2010).

<table>
<thead>
<tr>
<th>ISO week date</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>YYYYWwwD</td>
<td>2010W134</td>
</tr>
<tr>
<td>Extended</td>
<td>YYYY-Www-D</td>
<td>2010-W13-4</td>
</tr>
</tbody>
</table>

- **Ordinal date**  The last ISO date format is the ordinal date. The format is YYYY-DDD where
  YYYY is the year in the Gregorian calendar and DDD is the ordinal number of a calendar day within
  the calendar year. For example, '2010-091' represents the first day of April in 2010. ISO 8601 does not
  require the separator character. For example, '2010091' also represents April 1 2010. The maximum
  ordinal date is 366 for those years with leap years. For example, '2008366' represents the last day of
  the year in 2008 (December 31 2008).

<table>
<thead>
<tr>
<th>ISO ordinal date</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>YYYYYDDD</td>
<td>2010091</td>
</tr>
<tr>
<td>Extended</td>
<td>YYYY-DDD</td>
<td>2010-091</td>
</tr>
</tbody>
</table>
Other date formats are supported. SQL Anywhere is very flexible in its interpretation of strings containing dates. Whenever there is any ambiguity, the interpretation of the date value is guided by the date_order and nearest_century database option settings. For example, depending on the date_order setting, '02/05/2002' can be interpreted by the database server as the 2nd of May (DMY), or the 5th of February (MDY), or an illegal value (YMD).

The nearest_century setting determines whether a two-digit year value is interpreted as a year in the twentieth or twenty-first century. For example, in the string '02/05/10', the date_order setting would determine whether 02 or 10 is interpreted as the year and the nearest_century setting would determine whether 02 represented 1902 or 2002, or whether 10 represented 1910 or 2010. The value of the nearest_century option affects the interpretation of 2-digit years: 2000 is added to values less than nearest_century and 1900 is added to all other values. The default value of this option is 50. So, by default, the year 50 is interpreted as 1950 and the year 49 is interpreted as 2049.

The following table shows how the first day of April in 2010 could be specified using the indicated date_order setting and a nearest_century setting of 50.

<table>
<thead>
<tr>
<th>date_order</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>YMD</td>
<td>YYYY/MM/DD</td>
<td>2010/04/01</td>
</tr>
<tr>
<td>YMD</td>
<td>YY/MM/DD</td>
<td>10/04/01</td>
</tr>
<tr>
<td>MDY</td>
<td>MM/DD/YYYY</td>
<td>04/01/2010</td>
</tr>
<tr>
<td>MDY</td>
<td>MM/DD/YY</td>
<td>04/01/10</td>
</tr>
<tr>
<td>DMY</td>
<td>DD/MM/YYYY</td>
<td>01/04/2010</td>
</tr>
<tr>
<td>DMY</td>
<td>DD/MM/YY</td>
<td>01/04/10</td>
</tr>
</tbody>
</table>

Since ISO 8601 formats are not ambiguous and are not affected by the user’s setting of date_order and nearest_century, their use is recommended.

Dates can also be specified using month names. Examples are '2010 April 01', 'April 1, 2010', and '1 April 2010'. When the year is ambiguously specified, the date_order option is used to factor the year and day of month parts. Therefore, '01 April 10' is interpreted as April 10 2001 when the date_order is 'YMD' or as April 1 2010 when the date_order is 'DMY'.

The year in a date can range from 0001 to 9999. The minimum date in SQL Anywhere is 0001-01-01.

If a string contains only a partial date specification, default values are used to fill out the date. The following defaults are used:

- **year**  The current year is used when no year is specified (for example, 'April 1').
- **month** The current month is used when no year and month are specified (for example, '23:59:59') or 01 if a year is specified (for example, '2010').
- **day**  The current day is used when no year and month are specified (for example, '23:59:59') or 01 if a month is specified (for example, 'April').

In the following example, the date value is constructed from the current date.

```sql
SELECT CAST('23:59:59' AS TIMESTAMP);
```

**Time formats**

The time of day can be specified in the ISO 8601 format, using the 24-hour timekeeping system. It is hh:mm:ss, where hh is the number of complete hours that have passed since midnight, mm is the number of complete minutes since the start of the hour, and ss is the number of complete seconds since the start of the minute. For example, '23:59:59' represents the time one second before midnight.

The ISO 8601 standard allows for the omission of seconds and minutes. For example, '23:59' represents the time sixty seconds before midnight.

The ISO 8601 standard also allows you to include a decimal fraction to the seconds unit. Fractional seconds are specified using a comma (,) or a period (.). The fraction is stored to a maximum of six decimal places. For example, '23:59:59.500000' and '23:59:59.500000' both represent the time one-half second before midnight. SQL Anywhere does not support fractional minutes or hours.

ISO 8601 does not require the colon separator character when the time of day is included with a date specification. For example, '235959' represents the time one second before midnight.

The maximum time of day is '24:00:00'. It represents midnight. When combined with a date, it represents midnight, or 00:00:00 of the next day. For example, '2010-04-01 24:00:00' is equivalent to '2010-04-02 00:00:00'.

<table>
<thead>
<tr>
<th>ISO time</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic (with date)</td>
<td>hhmmss.ssssss</td>
<td>20100401 235959.500000</td>
</tr>
<tr>
<td>Basic (with date)</td>
<td>hhmmss,ssssss</td>
<td>20100401 235959,500000</td>
</tr>
<tr>
<td>Extended</td>
<td>hh:mm:ss.ssssss</td>
<td>23:59:59.500000</td>
</tr>
<tr>
<td>Extended</td>
<td>hh:mm:ss,ssssss</td>
<td>23:59:59,500000</td>
</tr>
</tbody>
</table>

The non-ISO AM and PM designators are also supported. For example, '11:59:59 PM' is equivalent to '23:59:59'.

<table>
<thead>
<tr>
<th>AM/PM</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>hh:mm:ss.ssssss AM</td>
<td>11:59:59.500000 AM</td>
</tr>
<tr>
<td>AM</td>
<td>hh:mm:ss,ssssss AM</td>
<td>11:59:59,500000 AM</td>
</tr>
<tr>
<td>PM</td>
<td>hh:mm:ss,ssssss PM</td>
<td>11:59:59,500000 PM</td>
</tr>
</tbody>
</table>
### AM/PM

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM</td>
<td>hh:mm:ss,ssssss PM</td>
</tr>
<tr>
<td>11:59:59.500000 PM</td>
<td></td>
</tr>
</tbody>
</table>

### Date with time formats

ISO 8601 permits the date and time of day to be combined using a space character or the letter `T`. For example, `'2010-04-01 23:59:59'` and `2010-04-01T23:59:59'` both represent the time one second before midnight on the first day of April in 2010. The hyphen and colon separator characters can be omitted. For example, `'20100401T235959'` also represents the same date and time. As an extension to this format, SQL Anywhere also supports the omission of the date and time separator. For example, `'20100401235959'` also represents the same date and time.

SQL Anywhere supports the mixing of basic and extended date and time formats. For example, `'20100401T23:59:59'` combines both the basic and extended formats.

### Time zone formats

ISO 8601 also permits the addition of a time zone offset to a date and time of day string. The format is one of:

- **Z** (Zulu) The date and time of day are in Coordinated Universal Time (UTC). For example, `'2010-04-01 23:00:00Z'` represents 11:00 PM Coordinated Universal Time on the first day of April in 2010.

- **+hh:mm** The specified date and time of day are the indicated number of hours and minutes ahead of UTC. For example, `'2010-04-01 23:00:00+04:00'` represents 11:00 PM on the first day of April in 2010 in a time zone 4 hours east of UTC.

- **-hh:mm** The specified date and time of day are the indicated number of hours and minutes behind UTC. For example, `'2010-04-01 23:00:00-05:00'` represents 11:00 PM on the first day of April in 2010 in a time zone 5 hours west of UTC.

If the minutes are 0, they do not need to be specified in the time zone offset. Also, a space can precede the time zone offset. For example, `'2010-04-01 23:00:00 -03:30'` represents 11:00 PM on the first day of April in 2010 in a time zone three and a half hours west of UTC.

### ISO time zone

<table>
<thead>
<tr>
<th>ISO time zone</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Z</td>
<td>20100401 235959Z</td>
</tr>
<tr>
<td>Basic</td>
<td>+hhmm</td>
<td>20100401 235959+0400</td>
</tr>
<tr>
<td>Basic</td>
<td>+hh</td>
<td>20100401 235959+04</td>
</tr>
<tr>
<td>Basic</td>
<td>-hhmm</td>
<td>20100401 235959-0500</td>
</tr>
<tr>
<td>Basic</td>
<td>-hh</td>
<td>20100401 235959-05</td>
</tr>
</tbody>
</table>
ISO time zone | Format            | Example                  |
-------------|-------------------|--------------------------|
Basic        | using T, fraction | 20100401T235959.50-0330  |
Extended     | Z                 | 2010-04-01 23:59:59Z     |
Extended     | +hh:mm            | 2010-04-01 23:59:59+04:00|
Extended     | -hh:mm            | 2010-04-01 23:59:59-05:00|
Extended     | using T, fraction | 2010-04-01T23:59:59.50-03:30|

SQL Anywhere supports the mixing of basic and extended date, time, and time zone formats. For example, '20100401T23:59:59-05' combines both basic and extended formats.

See also
- “Comparisons of dates and times” on page 134
- “DATE data type” on page 115
- “date_format option” [SQL Anywhere Server - Database Administration]
- “nearest_century option” [SQL Anywhere Server - Database Administration]
- “TIME data type” on page 120
- “TIMESTAMP data type” on page 121
- “TIMESTAMP WITH TIME ZONE data type” on page 123

## Retrieval of dates and times from the database

Dates and times can be retrieved from the database in one of the following ways:

- Using any interface, as a string
- Using ODBC or OLE DB, as a binary value (using an ODBC TIMESTAMP_STRUCT structure for example)
- Using embedded SQL, as a SQLDATETIME structure

Date and time of day with a time zone offset can be retrieved from the database as a string only.

When a date or time, with or without a time zone offset, is retrieved as a string, it is retrieved in the format specified by the database options date_format, time_format, timestamp_format, and timestamp_with_time_zone_format.

The following arithmetic operators are allowed on dates:

- `timestamp + integer` Add the specified number of days to a date or timestamp.
- `timestamp - integer` Subtract the specified number of days from a date or timestamp.
- `date - date` Compute the number of days between two dates or timestamps.
● **date + time**  Create a timestamp combining the given date and time.

**Leap years**

SQL Anywhere uses a globally accepted algorithm for determining which years are leap years. Using this algorithm, a year is considered a leap year if it is divisible by four, unless the year is a century date (such as the year 1900), in which case it is a leap year only if it is divisible by 400.

SQL Anywhere handles all leap years correctly. For example, the following SQL statement results in a return value of "Tuesday":

```
SELECT DAYNAME('2000-02-29');
```

SQL Anywhere accepts February 29, 2000—a leap year—as a date, and using this date determines the day of the week.

However, the following statement is rejected by SQL Anywhere:

This statement results in an error (cannot convert '2001-02-29' to a date) because February 29th does not exist in the year 2001.

**See also**

- “SET OPTION statement” on page 972
- “Date and time functions” on page 146

**DATE data type**

The DATE data type stores calendar dates, such as a year, month, and day.

**Syntax**

```
DATE
```

**Remarks**

The format in which DATE values are retrieved as strings by applications is controlled by the date_format option setting. For example, a DATE value representing the 19th of July, 2010 can be returned to an application as 2010/07/19, or as Jul 19, 2010 depending on the date_format option setting.

A DATE value requires 4 bytes of storage.
DATETIME data type

DATETIME stores date and time of day information.

Syntax

```
DATETIME
```

Remarks

DATETIME is a Transact-SQL type.

The format in which DATETIME values are retrieved as strings by applications is controlled by the timestamp_format option setting. For example, the DATETIME value 2010/04/01T23:59:59.999999 can be returned to an application as 2010/04/01 23:59:59, or as April 1, 2010 23:59:59.999999 depending on the timestamp_format option setting.

A DATETIME value requires 8 bytes of storage.

Although the range of possible dates for the DATETIME data type is the same as the DATE type (covering years 0001 to 9999), the useful range of the DATETIME date type is from 1600-02-28 23:59:59 to 7911-01-01 00:00:00. Before and after this range, the hours and minutes portion of the DATETIME value is not retained.
Note
When the precision of the DATETIME value is reduced, built-in functions that pertain to minutes or seconds will produce meaningless results.

When a DATETIME value is converted to a DATETIMEOFFSET, the connection's time_zone_adjustment setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a DATETIMEOFFSET value is converted to DATETIME, the offset is discarded.

See also
- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMP special value” on page 70
- “Date and time functions” on page 146
- “Ways to send dates and times to the database” on page 109
- “DATE function [Date and time]” on page 203
- “date_format option” [SQL Anywhere Server - Database Administration]
- “date_order option” [SQL Anywhere Server - Database Administration]
- “DATE data type” on page 115
- “DATETIME function [Date and time]” on page 209
- “DATETIMEOFFSET data type” on page 117
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP data type” on page 121
- “TIMESTAMP WITH TIME ZONE data type” on page 123
- “timestamp_format option” [SQL Anywhere Server - Database Administration]
- “nearest_century option” [SQL Anywhere Server - Database Administration]
- “UTC TIMESTAMP special value” on page 78

Standards and compatibility
- SQL/2008 Vendor extension.
- Transact-SQL DATETIME, rather than TIMESTAMP, is used by Adaptive Server Enterprise. The DATETIME type in Adaptive Server Enterprise supports dates between January 1, 1753 and December 31, 9999 and supports less precision with the time portion of the value. In SQL Anywhere, DATETIME is implemented as a TIMESTAMP without these restrictions. You should be aware of these differences when migrating data between SQL Anywhere and Adaptive Server Enterprise.

**DATETIMEOFFSET data type**

The DATETIMEOFFSET data type is an alias for TIMESTAMP WITH TIME ZONE, used to store date, time of day, and time zone information.

Syntax

```sql
DATETIMEOFFSET
```
Remarks

The DATETIMEOFFSET value contains the year, month, day, hour, minute, second, fraction of a second, and number of minutes before or after Coordinated Universal Time (UTC). The fraction is stored to 6 decimal places.

The format in which DATETIMEOFFSET values are retrieved as strings by applications is controlled by the timestamp_with_time_zone_format option setting. For example, the DATETIMEOFFSET value 2010/04/01T23:59:59.999999-6:00 can be returned to an application as 2010/04/01 23:59:59 -06:00, or as April 1, 2010 23:59:59.999999 -06:00 depending on the timestamp_with_time_zone_format option setting.

A DATETIMEOFFSET value requires 10 bytes of storage.

Although the range of possible dates for the DATETIMEOFFSET data type is the same as the DATE type (covering years 0001 to 9999), the useful range of DATETIMEOFFSET date types is from 1600-02-28 23:59:59 to 7911-01-01 00:00:00. Before and after this range, the hours and minutes portion of the DATETIMEOFFSET value is not retained.

Do not use DATETIMEOFFSET for computed columns or in materialized views because the value of the governing time_zone_adjustment option varies between connections based on their location and the time of year.

Two DATETIMEOFFSET values are considered identical when they represent the same instant in UTC, regardless of the TIME ZONE offset applied. For example, the following statement returns Yes because the results are considered identical:

```sql
IF CAST('2009-07-15 08:00:00 -08:00' AS DATETIMEOFFSET) =
CAST('2009-07-15 11:00:00 -05:00' AS DATETIMEOFFSET) THEN
SELECT 'Yes'
ELSE
SELECT 'No'
END IF;
```

If you omit the time zone offset from a DATETIMEOFFSET value, it defaults to the current UTC offset of the client regardless of whether the timestamp represents a date and time in standard time or daylight time. For example, if the client is located in the Eastern Standard time zone and executes the following statement while daylight time is in effect, then a timestamp with a time zone appropriate for the Atlantic Standard time zone (-4 hours from UTC) will be returned.

```sql
SELECT CAST('2009/01/30 12:34:55' AS DATETIMEOFFSET);
```

The comparison of DATETIMEOFFSET values with timestamps without time zones is not recommended because the default time zone offset of the client varies with the geographic location of the client and with the time of the year.

Execute the following statement to determine the current time zone offset in minutes for a client:

```sql
SELECT CONNECTIONPROPERTY( 'TimeZoneAdjustment' );
```

Note
The TimeZoneAdjustment connection property is not supported in UltraLite databases.
See also

- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMP special value” on page 70
- “Date and time functions” on page 146
- “Ways to send dates and times to the database” on page 109
- “DATE data type” on page 115
- “DATETIME data type” on page 116
- “DATE function [Date and time]” on page 203
- “DATETIME function [Date and time]” on page 209
- “Expressions” on page 21
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP special value” on page 76
- “TIMESTAMP data type” on page 121
- “TIMESTAMP WITH TIME ZONE data type” on page 123
- “timestamp_with_time_zone_format option” [SQL Anywhere Server - Database Administration]
- “nearest_century option” [SQL Anywhere Server - Database Administration]
- “date_order option” [SQL Anywhere Server - Database Administration]
- “UTC TIMESTAMP special value” on page 78

Standards and compatibility

- SQL/2008 The specific use of DATETIMEOFFSET is a vendor extension. To be compatible with SQL/2008, use TIMESTAMP WITH TIME ZONE. The TIMESTAMP WITH TIME ZONE type is optional SQL language feature F411 of the SQL/2008 standard.

SMALLDATETIME data type

SMALLDATETIME is a domain, implemented as TIMESTAMP, used to store date and time of day information. SMALLDATETIME is a Transact-SQL type.

Syntax

SMALLDATETIME

Remarks

For information about the specification of dates and time of day, see “Ways to send dates and times to the database” on page 109.
Standards and compatibility

- SQL/2008  Vendor extension.
- Transact-SQL  SMALLDATETIME is supported by Adaptive Server Enterprise. In Adaptive Server Enterprise, the SMALLDATETIME type supports dates between January 1, 1900 and June 6, 2079 and supports less precision with the time portion of the value. In SQL Anywhere, SMALLDATETIME is implemented as a TIMESTAMP without these restrictions. You should be aware of these differences when migrating data between SQL Anywhere and Adaptive Server Enterprise.

**TIME data type**

The TIME data type stores the time of day, containing the hour, minute, second, and fraction of a second.

**Syntax**

```plaintext
TIME
```

**Remarks**

The format in which TIME values are retrieved as strings by applications is controlled by the time_format option setting. For example, the TIME value `23:59:59.999999` can be returned to an application as `23:59:59`, `23:59:59.999`, or `23:59:59.999999` depending on the time_format option setting.

A TIME value requires 8 bytes of storage.

When using ODBC, a TIME value sent or retrieved as a binary value (using an ODBC TIME_STRUCT structure) is restricted to an accuracy of hours, minutes, and seconds. Fractional seconds are not part of
the structure. For this reason, TIME values should be sent or retrieved as strings if increased accuracy is desired.

See also
- “Time formats” on page 112
- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMP special value” on page 70
- “Date and time functions” on page 146
- “DATE data type” on page 115
- “DATETIME data type” on page 116
- “DATE function [Date and time]” on page 203
- “DATETIME function [Date and time]” on page 209
- “Expressions” on page 21
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIMESTAMP special value” on page 76
- “TIMESTAMP data type” on page 121
- “TIMESTAMP WITH TIME ZONE data type” on page 123
- “timestamp_format option” [SQL Anywhere Server - Database Administration]
- “timestamp_with_time_zone_format option” [SQL Anywhere Server - Database Administration]
- “UTC TIMESTAMP special value” on page 78

Standards and compatibility
- Transact-SQL  The TIME data type is supported by Adaptive Server Enterprise. However, Adaptive Server Enterprise supports millisecond resolution (three digits) rather than microsecond resolution (six digits). You should be aware of these differences when migrating data between SQL Anywhere and Adaptive Server Enterprise. To migrate TIME values, use the Adaptive Server Enterprise BIGTIME data type.

**TIMESTAMP data type**

The TIMESTAMP data type stores a point in time containing the year, month, day, hour, minute, second, and fraction of a second stored to six decimal places.

Syntax

```
TIMESTAMP
```

Remarks

The format in which TIMESTAMP values are retrieved as strings by applications is controlled by the timestamp_format option setting. For example, the TIMESTAMP value 2010/04/01T23:59:59.999999
can be returned to an application as 2010/04/01 23:59:59 or as April 1, 2010 23:59:59.999999, depending on the timestamp_format option setting.

A TIMESTAMP value requires 8 bytes of storage.

Although the range of possible dates for the TIMESTAMP data type is the same as the DATE type (covering years 0001 to 9999), the useful range of TIMESTAMP date types is from 1600-02-28 23:59:59 to 7911-01-01 00:00:00. Before and after this range, the hours and minutes portion of the TIMESTAMP value is not retained.

**Note**

> When the precision of the TIMESTAMP value is reduced, built-in functions that pertain to minutes or seconds produce meaningless results.

When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the connection's time_zone_adjustment setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a TIMESTAMP WITH TIME ZONE value is converted to TIMESTAMP, the offset is discarded.

**See also**

- “Ways to send dates and times to the database” on page 109
- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMP special value” on page 70
- “Date and time functions” on page 146
- “DATE data type” on page 115
- “DATETIME data type” on page 116
- “DATE function [Date and time]” on page 203
- “DATETIME function [Date and time]” on page 209
- “date_order option” [SQL Anywhere Server - Database Administration]
- “Expressions” on page 21
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “nearest_century option” [SQL Anywhere Server - Database Administration]
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP special value” on page 76
- “TIMESTAMP WITH TIME ZONE data type” on page 123
- “timestamp_format option” [SQL Anywhere Server - Database Administration]
- “timestamp_with_time_zone_format option” [SQL Anywhere Server - Database Administration]
- “UTC TIMESTAMP special value” on page 78

**Standards and compatibility**

- **SQL/2008**  Compatible with SQL/2008.
- **Transact-SQL**  Adaptive Server Enterprise uses the DATETIME type for TIMESTAMP values.
TIMESTAMP WITH TIME ZONE data type

The TIMESTAMP WITH TIME ZONE data type stores a point in time with a time zone offset.

Syntax

TIMESTAMP WITH TIME ZONE

Remarks

The TIMESTAMP WITH TIME ZONE value contains the year, month, day, hour, minute, second, fraction of a second, and number of minutes before or after Coordinated Universal Time (UTC). The fraction is stored to six decimal places.

The format in which TIMESTAMP WITH TIME ZONE values are retrieved as strings by applications is controlled by the timestamp_with_time_zone_format option setting. For example, the TIMESTAMP WITH TIME ZONE value 2010/04/01T23:59:59.999999-6:00 can be returned to an application as 2010/04/01 23:59:59 -06:00 or as April 1, 2010 23:59:59.999999 -06:00, depending on the timestamp_with_time_zone_format option setting.

A TIMESTAMP WITH TIME ZONE value requires 10 bytes of storage.

Although the range of possible dates for the TIMESTAMP WITH TIME ZONE data type is the same as the DATE type (covering years 0001 to 9999), the useful range of TIMESTAMP WITH TIME ZONE date types is from 1600-02-28 23:59:59 to 7911-01-01 00:00:00. Before and after this range, the hours and minutes portion of the TIMESTAMP WITH TIME ZONE value is not retained.

Do not use TIMESTAMP WITH TIME ZONE for computed columns or in materialized views because the value of the governing time_zone_adjustment option varies between connections based on their location and the time of year.

Two TIMESTAMP WITH TIME ZONE values are considered identical when they represent the same instant in UTC, regardless of the TIME ZONE offset applied. For example, the following statement returns Yes because the results are considered identical:

```sql
IF CAST('2009-07-15 08:00:00 -08:00' AS TIMESTAMP WITH TIME ZONE) =
    CAST('2009-07-15 11:00:00 -05:00' AS TIMESTAMP WITH TIME ZONE) THEN
    SELECT 'Yes'
ELSE
    SELECT 'No'
END IF;
```

If you omit the time zone offset from a TIMESTAMP WITH TIME ZONE value, it defaults to the current UTC offset of the client regardless of whether the timestamp represents a date and time in standard time or daylight time. For example, if the client is located in the Eastern Standard time zone and executes the following statement while daylight time is in effect, then a timestamp with a time zone appropriate for the Atlantic Standard time zone (−4 hours from UTC) is returned.

```sql
SELECT CAST('2009/01/30 12:34:55' AS TIMESTAMP WITH TIME ZONE)
```

- **Comparing TIMESTAMP WITH TIME ZONE with other data types** The comparison of TIMESTAMP WITH TIME ZONE values with timestamps without time zones is not recommended because the default time zone offset of the client varies with the geographic location of the client and with the time of the year.
Execute the following statement to determine the current time zone offset in minutes for a client:

```
SELECT CONNECTION_PROPERTY( 'TimeZoneAdjustment' );
```

- **Converting to or from TIMESTAMP WITH TIME ZONE**: When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the connection's time_zone_adjustment setting is used for the time zone offset in the result. In other words, the value is considered to be local to the connection. When a TIMESTAMP WITH TIME ZONE value is converted to TIMESTAMP, the offset is discarded. Conversions to or from types other than strings, date, or date-time types is not supported.

**See also**

- “Comparisons of dates and times” on page 134
- “Ways to send dates and times to the database” on page 109
- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMP special value” on page 70
- “Date and time functions” on page 146
- “DATE data type” on page 115
- “DATETIME data type” on page 116
- “DATE function [Date and time]” on page 203
- “DATETIME function [Date and time]” on page 209
- “DATETIMEOFFSET data type” on page 117
- “date_order option” [SQL Anywhere Server - Database Administration]
- “Expressions” on page 21
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “nearest_century option” [SQL Anywhere Server - Database Administration]
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP data type” on page 121
- “TIMESTAMP special value” on page 76
- “TIMESTAMP data type” on page 121
- “timestamp_format option” [SQL Anywhere Server - Database Administration]
- “timestamp_with_time_zone_format option” [SQL Anywhere Server - Database Administration]
- “UTC TIMESTAMP special value” on page 78

**Standards and compatibility**

- **SQL/2008**: Support for TIMESTAMP WITH TIME ZONE is optional SQL language feature F411 of the SQL/2008 standard.

**Binary data types**

Binary data types store binary data, including images and other types of information that are not interpreted by the database.
**BINARY data type**

The BINARY data type stores binary data of a specified maximum length (in bytes).

**Syntax**

```
BINARY [ ( max-length ) ]
```

**Parameters**

- `max-length`  The maximum length of the value, in bytes. If the length is not specified, then it is 1.

  The length must be in the 1 to 32767 range.

**Remarks**

During comparisons, BINARY values are compared exactly byte for byte. This differs from the CHAR data type, where values are compared using the collation sequence of the database.

If one binary string is a prefix of the other, the shorter string is considered to be less than the longer string.

Unlike CHAR values, BINARY values are not transformed during character set conversion.

BINARY is semantically equivalent to VARBINARY. It is a variable-length type. In other database management systems, BINARY is a fixed-length type.

**See also**

- “VARBINARY data type” on page 127
- “LONG BINARY data type” on page 126
- “String functions” on page 152
- “Bitwise operators” on page 19

**Standards and compatibility**

- **SQL/2008**  The BINARY data type is SQL language feature T021 of the SQL/2008 standard.

**IMAGE data type**

The IMAGE data type stores binary data of arbitrary length.

**Syntax**

```
IMAGE
```

**Remarks**

IMAGE is a domain, implemented as LONG BINARY.

**See also**

- “LONG BINARY data type” on page 126
- “String functions” on page 152
Standards and compatibility
● SQL/2008  Vendor extension.

LONG BINARY data type
The LONG BINARY data type stores binary data of arbitrary length.

Syntax
LONG BINARY

Remarks
The maximum size is 2 GB minus 1 byte (\(2^{31} - 1\)).

See also
● “BINARY data type” on page 125
● “VARBINARY data type” on page 127

Standards and compatibility
● SQL/2008  The LONG BINARY data type comprises SQL language features T021, "BINARY and VARBINARY data types", and T041, "Basic LOB data type support" in the SQL/2008 standard.

UNIQUEIDENTIFIER data type
The UNIQUEIDENTIFIER data type stores UUID (also known as GUID) values.

Syntax
UNIQUEIDENTIFIER

Remarks
The UNIQUEIDENTIFIER data type is typically used for a primary key or other unique column to hold UUID (Universally Unique Identifier) values that uniquely identify rows. The NEWID function generates UUID values in such a way that a value produced on one computer does not match a UUID produced on another computer. UNIQUEIDENTIFIER values generated using NEWID can therefore be used as keys in a synchronization environment.

For example:

```sql
CREATE TABLE T1 (  
  pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),  
  c1 INT );
```

UUID values are also referred to as GUID (Globally Unique Identifier) values. UUID values contain hyphens so they are compatible with other RDBMSs. You can change this by setting the uuid_has_hyphens option to Off.

UNIQUEIDENTIFIER values are automatically converted between string and binary values as needed.
UNIQUEIDENTIFIER values are stored as BINARY(16) but are described to client applications as BINARY(36). This description ensures that if the client fetches the value as a string, it has allocated enough space for the result. For ODBC client applications, uniqueidentifier values appear as a SQL_GUID type.

See also

- “The NEWID default” [SQL Anywhere Server - SQL Usage]
- “NEWID function [Miscellaneous]” on page 303
- “UUIDTOSTR function [String]” on page 401
- “STRTOUUID function [String]” on page 377
- “uuid_has_hyphens option” [SQL Anywhere Server - Database Administration]
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

VARBINARY data type

The VARBINARY data type stores binary data of a specified maximum length (in bytes).

Syntax

```
VARBINARY [ ( max-length ) ]
```

Parameters

- **max-length**  The maximum length of the value, in bytes. If the length is not specified, then it is 1.
  
The length must be in the 1 to 32767 range.

Remarks

During comparisons, VARBINARY values are compared exactly byte for byte. This behavior differs from the CHAR data type, where values are compared using the collation sequence of the database.

If one binary string is a prefix of the other, the shorter string is considered to be less than the longer string.

VARBINARY values are not transformed during character set conversion.

VARBINARY can also be specified as BINARY VARYING. Regardless of which syntax is used, the data type is described as VARBINARY.

See also

- “BINARY data type” on page 125
- “LONG BINARY data type” on page 126
- “String functions” on page 152
- “Bitwise operators” on page 19
Standards and compatibility

- **SQL/2008**  The VARBINARY data type comprises SQL language feature T021, "BINARY and VARBINARY data types" in the SQL/2008 standard.

Composite data types

Composite data types are values that are comprised of zero or more elements, where each element has a value of a particular data type.

SQL Anywhere supports the ROWS and ARRAYS composite data types. These data types are a more efficient way to store lists because they provide the ability to define the structure and data type of their values. They also make access to list elements easier to achieve, either directly, by using double square brackets; or as result set, by using the UNNEST operator. Consider using the ARRAY data type if you are storing lists as delimited strings in VARCHAR columns, and parsing them using sa_split_list. ARRAYS are very helpful when storing different objects that are all related in some ways. ROWS are helpful when storing multiple values related to one object.

Declaring an ARRAY type

An ARRAY type is a homogeneous, ordered collection that can be passed in whole or in part as an argument to SQL stored procedures or functions.

An ARRAY type can consist of up to 6.4 million elements. An ARRAY is initialized to a zero-length ARRAY of the declared type, where each element is NULL.

An ARRAY constructor builds an ARRAY value so the array can be processed in a query or passed as an argument to a SQL stored procedure or function.

The database server supports the following syntaxes for declaring variables of the ARRAY type:

1. **DECLARE variable-name element-type-name ARRAY [(maximum-size)]**
2. **DECLARE variable-name ARRAY [(maximum-size)] OF element-type-name**

The array can contain up to 6.4 million elements if maximum-size is omitted. Variables of the ARRAY type are not initialized to NULL the way all other non-composite variables are. Instead, they are initialized to a zero-length array.

ARRAY types cannot be stored as columns in a base or temporary table and are not supported in:

- the outermost SELECT list of a view definition
- a top-level SELECT block or query expression that is returned to the client
- a base table
- a temporary table
- an embedded SQL FETCH statement

The following example illustrates how to declare an array of 5 integers:

```
DECLARE NewArray INTEGER ARRAY(5);
```
The following example illustrates an alternative way of declaring an array of 5 integers that is compatible with the Oracle syntax:

```
DECLARE NewArray ARRAY(5) OF INTEGER;
```

The following example illustrates how to declare a two-dimensional array, where New2DArray contains 10 elements, each of which is a five-element array of integers:

```
DECLARE New2DArray INTEGER ARRAY(5) ARRAY(10);
```

The following example illustrates an alternative way of declaring a two-dimensional array that is compatible with the Oracle syntax, where New2DArray contains 10 elements, each of which is a five-element array of integers:

```
DECLARE New2DArray ARRAY(10) OF ARRAY(5) OF INTEGER;
```

### Declaring a ROW type

A ROW type is described by a row type descriptor, which consists of the field descriptor of each field in the ROW type. ROW types are restricted to 45000 fields, which is the same limit as the number of columns in a table.

Variables in the ROW type are initialized to a ROW of the declared type where each field is initialized as NULL.

A ROW supports the construction of structured types, consisting of a group of fields of potentially different types. ROW types can be part of higher-order row types, which permits complex structures involving other row types and arrays.

The following example illustrates how to declare a variable, student, that is defined as a structured type of four different fields:

```
DECLARE student ROW(
  studentID INTEGER,
  student_first_name VARCHAR(40),
  student_last_name VARCHAR(50),
  student address LONG VARCHAR);
```

The following example illustrates how to assign a ROW as a complete structure:

```
DECLARE employee ROW(
  empID INTEGER,
  address ROW(street_address LONG VARCHAR, city VARCHAR(50), province VARCHAR(30), country VARCHAR(40))
);
DECLARE temp_address ROW(street_address LONG VARCHAR, city VARCHAR(50), province VARCHAR(30), country VARCHAR(40))
SET temp_address = employee.address;
```
Spatial data types

SQL Anywhere supports many spatial data types. The documentation for these data types are located with the spatial SQL API documentation.

See also
- “Supported spatial data types and their hierarchy” [SQL Anywhere Server - Spatial Data Support]

Domains

Domains are aliases for built-in data types, including precision and scale values where applicable, and optionally including DEFAULT values and CHECK conditions. Some domains, such as the monetary data types, are predefined in SQL Anywhere, but you can add more of your own.

Domains, also called user-defined data types, allow columns throughout a database to be automatically defined on the same data type, with the same NULL or NOT NULL condition, with the same DEFAULT setting, and with the same CHECK condition. Domains encourage consistency throughout the database and can eliminate some types of errors.

Simple domains

Domains are created using the CREATE DOMAIN statement.

The following statement creates a data type named street_address, which is a 35-character string.

```
CREATE DOMAIN street_address CHAR(35);
```

CREATE DATATYPE can be used as an alternative to CREATE DOMAIN, but is not recommended.

You must have the CREATE DATATYPE or CREATE ANY OBJECT system privilege to create domains. Once a data type is created, the user ID that executed the CREATE DOMAIN statement is the owner of that data type. Any user can use the data type. Unlike with other database objects, the owner name is never used to prefix the data type name.

The street_address data type can be used in exactly the same way as any other data type when defining columns. For example, the following table with two columns has the second column as a street_address column:

```
CREATE TABLE twocol (
    ID INT,
    street street_address
);
```

You can also drop a domain:
DROP DOMAIN street_address;

This statement can be executed only if the data type is not used in any table in the database. If you attempt to drop a domain that is in use, an error message appears.

See also

- “CREATE DOMAIN statement” on page 562
- “DROP DOMAIN statement” on page 756

Constraints and defaults with domains

Many of the attributes associated with columns, such as allowing NULL values, having a DEFAULT value, and so on, can be built into a domain. Any column that is defined on the data type automatically inherits the NULL setting, CHECK condition, and DEFAULT values. This allows uniformity to be built into columns with a similar meaning throughout a database.

For example, many primary key columns in the SQL Anywhere sample database are integer columns holding ID numbers. The following statement creates a data type that can be useful for such columns:

```sql
CREATE DOMAIN ID INT
    NOT NULL
    DEFAULT AUTOINCREMENT
    CHECK( @col > 0 );
```

By default, a column created using the id data type does not allow NULLs, defaults to an auto-incremented value, and must hold a positive number. Any identifier could be used instead of \texttt{col} in the \texttt{@col} variable.

The attributes of a data type can be overridden by explicitly providing attributes for the column. A column created using the id data type with NULL values explicitly allowed does allow NULLs, regardless of the setting in the id data type.

Compatibility

- **Named constraints and defaults** In SQL Anywhere, domains are created with a base data type, and optionally a NULL or NOT NULL condition, a default value, and a CHECK condition. Named constraints and named defaults are not supported.

- **Creating data types** In SQL Anywhere, you can use the \texttt{sp_addtype} system procedure to add a domain, or you can use the \texttt{CREATE DOMAIN} statement.

Data type comparisons

When a comparison (such as =) is performed between arguments with different data types, one or more arguments must be converted so that the comparison operation is done using one data type.

Some rules may lead to conversions that fail, or lead to unexpected results from the comparison. In these cases, you should explicitly convert one of the arguments using \texttt{CAST} or \texttt{CONVERT}.

You can override these conversion rules by explicitly casting arguments to another type. For example, to compare a DATE and a CHAR as a CHAR, explicitly cast the DATE to a CHAR.
Lossy conversion and substitution characters

When a character cannot be represented in the character set into which it is being converted, a substitution character is used instead. Conversions of this type are considered **lossy**; the original character is lost if it cannot be represented in the destination character set.

Also, not only may different character sets have a different substitution character, but the substitution character for one character set can be a non-substitution character in another character set. This is important to understand when multiple conversions are performed on a character because the final character may not appear as the expected substitution character of the destination character set.

For example, suppose that the client character set is Windows-1252, and the database character set is ISO_8859-1:1987, the U.S. default for some versions of Unix. Then, suppose a non-Unicode client application (for example, embedded SQL) attempts to insert the euro symbol into a CHAR, VARCHAR, or LONG VARCHAR column. Since the character does not exist in the CHAR character set, the substitution character for ISO_8859-1:1987, 0x1A, is inserted.

Now, if this same ISO_8859-1:1987 substitution character is then fetched as Unicode (for example, by doing a `SELECT * FROM t` into a SQL_C_WCHAR bound column in ODBC), this character becomes the Unicode code point U+001A. (In Unicode the code point U+001A is the record separator control character.) However, the substitution character for Unicode is the code point U+FFFD. This example illustrates that even if your data contains substitution characters, those characters, due to multiple conversions, may not be converted to the substitution character of the destination character set.

Therefore, it is important to understand and test how substitution characters are used when converting between multiple character sets.

The `on_charset_conversion_failure` option can help determine the behavior during conversion when a character cannot be represented in the destination character set.

See also

- “Data type conversions” on page 137
- “Comparisons between CHAR and NCHAR” on page 132
- “on_charset_conversion_failure option” [SQL Anywhere Server - Database Administration]

Comparisons between CHAR and NCHAR

When a comparison is performed between a value of CHAR type (CHAR, VARCHAR, LONG VARCHAR) and a value of NCHAR type (NCHAR, NVARCHAR, LONG NVARCHAR), SQL Anywhere uses inference rules to determine the type in which the comparison should be performed. Generally, if one value is based on a column reference and the other is not, the comparison is performed in the type of the value containing the column reference.
The inference rules revolve around whether a value is based on a column reference. In the case where one value is a variable, a host variable, a literal constant, or a complex expression not based on a column reference and the other value is based on a column reference, then the constant-based value is implicitly cast to the type of the column-based value.

Following are the inference rules, in the order in which they are applied:

- If the NCHAR value is based on a column reference, the CHAR value is implicitly cast to NCHAR, and the comparison is done as NCHAR. This includes the case where both the NCHAR and CHAR value are based on column references.

- Else if the NCHAR value is not based on a column reference, and the CHAR value is based on a column reference, the NCHAR value is implicitly cast to CHAR, and the comparison is done as CHAR.

  It is important to consider the setting for the on_charset_conversion_failure option if you anticipate NCHAR to CHAR conversions since this option controls behavior if an NCHAR character cannot be represented in the CHAR character set.

- Else if neither value is based on a column reference, then the CHAR value is implicitly cast to NCHAR and the comparison is done as NCHAR.

**Examples**

The condition `Employees.GivenName = N'Susan'` compares a CHAR column (Employees.GivenName) to the literal N'Susan'. The value N'Susan' is cast to CHAR, and the comparison is performed as if it had been written as:

```sql
Employees.GivenName = CAST( N'Susan' AS CHAR );
```

Alternatively, the condition `Employees.GivenName = T.nchar_column` would find that the value T.nchar_column cannot be cast to CHAR. The comparison would be performed as if it were written as follows, and an index on Employees.GivenName cannot be used:

```sql
CAST( Employees.GivenName AS NCHAR ) = T.nchar_column;
```

**See also**

- “NCHAR to CHAR conversions” on page 138
- “NCHAR to CHAR conversions” on page 138
- “Lossy conversion and substitution characters” on page 132
- “CAST function [Data type conversion]” on page 174
- “CONVERT function [Data type conversion]” on page 187
- “CAST function [Data type conversion]” on page 174
- “on_charset_conversion_failure option” [SQL Anywhere Server - Database Administration]

**Comparisons between numeric data types**

SQL Anywhere uses the following rules when comparing numeric data types. The rules are examined in the order listed, and the first rule that applies is used:

1. If one argument is TINYINT and the other is INTEGER, convert both to INTEGER and compare.
2. If one argument is TINYINT and the other is SMALLINT, convert both to SMALLINT and compare.

3. If one argument is UNSIGNED SMALLINT and the other is INTEGER, convert both to INTEGER and compare.

4. If the data types of the arguments have a common super type, convert to the common super type and compare. The super types are the final data type in each of the following lists:

   - BIT » TINYINT » UNSIGNED SMALLINT » UNSIGNED INTEGER » UNSIGNED BIGINT » NUMERIC
   - SMALLINT » INTEGER » BIGINT » NUMERIC
   - REAL » DOUBLE
   - CHAR » LONG VARCHAR
   - BINARY » LONG BINARY

For example, if the two arguments are of types BIT and TINYINT, they are converted to NUMERIC.

**Comparisons of dates and times**

The table below summarizes the conversions that are implicit when comparing certain data types with date, time, or date-time data types.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Data type</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>DATE</td>
<td>CHAR cast to TIMESTAMP; DATE cast to TIMESTAMP</td>
</tr>
<tr>
<td>CHAR</td>
<td>TIME</td>
<td>CHAR cast to TIME</td>
</tr>
<tr>
<td>CHAR</td>
<td>TIMESTAMP</td>
<td>CHAR cast to TIMESTAMP</td>
</tr>
<tr>
<td>CHAR</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>CHAR cast to TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>DATE</td>
<td>TIME</td>
<td>illegal</td>
</tr>
<tr>
<td>DATE</td>
<td>TIMESTAMP</td>
<td>DATE cast to TIMESTAMP</td>
</tr>
<tr>
<td>DATE</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>DATE cast to TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>DATE</td>
<td>SMALLINT, INTEGER, BIGINT, and NUMERIC</td>
<td>SMALLINT, INTEGER, BIGINT, and NUMERIC value treated as a date string and cast to TIMESTAMP; DATE cast to TIMESTAMP</td>
</tr>
</tbody>
</table>
## Data type comparisons

<table>
<thead>
<tr>
<th>Data type</th>
<th>Data type</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>REAL, FLOAT, and DOUBLE</td>
<td>REAL, FLOAT, and DOUBLE treated as a number of days since 0000-02-29 and cast to TIMESTAMP; DATE cast to TIMESTAMP</td>
</tr>
<tr>
<td>TIME</td>
<td>TIMESTAMP</td>
<td>TIMESTAMP cast to TIME</td>
</tr>
<tr>
<td>TIME</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>illegal</td>
</tr>
<tr>
<td>TIME-STEMP</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>TIMESTAMP cast to TIMESTAMP WITH TIME ZONE</td>
</tr>
<tr>
<td>TIME-STEMP</td>
<td>SMALLINT, INTEGER, BIGINT, and NUMERIC</td>
<td>SMALLINT, INTEGER, BIGINT, and NUMERIC value treated as a date string and cast to TIMESTAMP</td>
</tr>
<tr>
<td>TIME-STEMP</td>
<td>REAL, FLOAT, and DOUBLE</td>
<td>REAL, FLOAT, and DOUBLE treated as a number of days since 0000-02-29 and cast to TIMESTAMP</td>
</tr>
</tbody>
</table>

The following points expand on the information presented in the table above.

1. Only values of type TIME, TIMESTAMP, and CHAR can be compared to a value of type TIME. Comparison with values of other data types results in a conversion error. When comparing a time value and a value of another type, the comparison data type is TIME.

2. When comparing a TIMESTAMP, SMALLINT, INTEGER, BIGINT, NUMERIC, REAL, FLOAT, or DOUBLE value to a DATE value, the comparison data type is always TIMESTAMP.

3. When comparing a TIMESTAMP WITH TIME ZONE value to a DATE value, the comparison data type is always TIMESTAMP WITH TIME ZONE.

4. When a time value is cast to a TIMESTAMP, the result is formed by combining the current date with the time value.

5. Exact numeric values of type SMALLINT, INTEGER, BIGINT, and NUMERIC can be converted to date values. The conversion is performed by treating the number as a string. For example, the integer value 20100401 represents the first day of April in 2010.

6. The unsigned exact numeric types BIT, TINYINT, UNSIGNED SMALLINT, UNSIGNED INTEGER, and UNSIGNED BIGINT cannot be converted to date values.

7. Approximate numeric values of type REAL, FLOAT, and DOUBLE can be converted to dates by treating the number as the number of days since the fictional date 0000-02-29. For example, 307 represents 0001-01-01 and 734169 represents 2010-04-01.

### See also

- “Date and time data types” on page 108
Comparisons of composite types

Array elements are compared starting from the first element. When a difference is found, the comparison stops and the result of the comparison between the most recently compared elements is returned. If all of the elements compare equal, then the arrays are equal. The comparisons performed are equivalent to those performed on expressions that are not held in arrays. If one array is shorter than another, and all elements of the shorter array are equal to the same elements of the longer array, the shorter array is considered less than the longer array.

When comparing arrays, the arrays must hold values with union-compatible data types. Duplicate elimination and GROUP BY are also supported over array expressions. For example, with the following array comparison, the query returns 1:

```sql
SELECT IF ARRAY(3,4,5) > ARRAY(2,3,4) THEN 1 ELSE 0 ENDIF;
```

Row types can be compared, used in joins, duplicate elimination, and grouping. Consider two row types similar to the row expression sample above:

```sql
DECLARE test1 ROW(x INT, w ROW(y INT, z INT));
DECLARE test2 ROW(a INT, b ROW(c INT, d CHAR(3)));
SET test1 = ROW(3, ROW(6,7));
SET test2 = ROW(3, ROW(8,'7'));
SELECT (IF (test1 > test2) THEN 1 ELSE 0 ENDIF) AS RESULT FROM dummy;
```

Two row expressions can be compared only if their structures match. However, while the row expressions must have the same structure, the names of the attributes of a row type do not need to be identical, and the data types of the individual leaf values do not need to be identical—only union compatible.

All ROW comparisons other than equality and inequality operations result in UNKNOWN.

Transact-SQL string to date/time conversions

If a string containing only a time value (no date) is converted to a date/time data type, SQL Anywhere uses the current date.

If the fraction portion of a time is less than 3 digits, SQL Anywhere interprets the value the same way regardless of the whether it is preceded by a period or a colon: one digit means tenths, two digits mean hundredths, and three digits mean thousandths.

Examples

SQL Anywhere converts the milliseconds value in the same manner regardless of the separator.

- 12:34:56.7 to 12:34:56.700
- 12:34:56:7 to 12:34:56.700
- 12.34.56.78 to 12:34:56.780
- 12.34.56:78 to 12:34:56.780
- 12:34:56.789 to 12:34:56.789
- 12:34:56:789 to 12:34:56.789
Other comparisons

1. If the data types are a mixture of CHAR (such as CHAR, VARCHAR, LONG VARCHAR, and so on, but not NCHAR types), convert to LONG VARCHAR and compare.

2. If the data type of any argument is UNIQUEIDENTIFIER, convert to UNIQUEIDENTIFIER and compare.

3. If the data type of any argument is a bit array (VARBIT or LONG VARBIT), convert to LONG VARBIT and compare.

4. If one argument has CHARACTER data type and the other has BINARY data type, convert to BINARY and compare.

5. If one argument is a CHAR type, and the other argument is an NCHAR type, use predefined inference rules.

6. If no rule exists, convert to NUMERIC and compare.

For example, if the two arguments have REAL and CHAR data types, they are both converted to NUMERIC.

See also

- “Comparisons between CHAR and NCHAR” on page 132

Data type conversions

Type conversions can happen automatically, or they can be explicitly requested using the CAST or CONVERT function. The following functions can also be used to force type conversions:

- **DATE function** Converts the expression into a DATE, and removes any hours, minutes or seconds. Conversion errors can be reported.

- **DATETIME function** Converts the expression into a TIMESTAMP, and removes any time zone. Conversion errors can be reported.

- **STRING function** This function is equivalent to CAST(value AS LONG VARCHAR).

- **VALUE+0.0** Equivalent to CAST( value AS DECIMAL ).

The following list is a high-level view of automatic data type conversions:

- If a string is used in a numeric expression or as an argument to a function that expects a numeric argument, the string is converted to a number.

- If a number is used in a string expression or as a string function argument, it is converted to a string before being used.

- All date constants are specified as strings. The string is automatically converted to a date before use.
There are certain cases where the automatic database conversions are not appropriate. For example, the automatic data type conversion fails in the example below.

```sql
'12/31/90' + 5
'a' > 0
```

See also

- “Data type conversion functions” on page 146
- “DATE function [Date and time]” on page 203
- “DATETIME function [Date and time]” on page 209
- “STRING function [String]” on page 376
- “CAST function [Data type conversion]” on page 174

**NCHAR to CHAR conversions**

NCHAR to CHAR conversions can occur as part of a comparison of CHAR and NCHAR data, or when specifically requested. This type of conversion is lossy because depending on the CHAR character set, there can be some NCHAR characters that cannot be represented in the CHAR type. When an NCHAR character cannot be converted to CHAR, a substitution character from the CHAR character set is used instead. For single-byte character sets, this is usually hex 1A.

Depending on the setting of the on_charset_conversion_failure option, when a character cannot be converted, one of the following can happen:

- a substitute character is used, and no warning is issued
- a substitute character is used, and a warning is issued
- an error is returned

Therefore, it is important to consider this option when converting from NCHAR to CHAR.

See also

- “Comparisons between CHAR and NCHAR” on page 132
- “on_charset_conversion_failure option” [SQL Anywhere Server - Database Administration]

**NULL constant conversions to NUMERIC and string types**

When converting a NULL constant to a NUMERIC, or to a string type such as CHAR, VARCHAR, LONG VARCHAR, BINARY, VARBINARY, and LONG BINARY the size is set to 0. For example:

```sql
SELECT CAST( NULL AS CHAR ) returns CHAR(0)
SELECT CAST( NULL AS NUMERIC ) returns NUMERIC(1,0)
```
Bit array conversions

Converting integers to bit arrays

When converting an integer to a bit array, the length of the bit array is the number of bits in the integer type, and the bit array's value is the binary representation. The most significant bit of the integer becomes the first bit of the array.

Examples

SELECT CAST( CAST( 1 AS BIT ) AS VARBIT ) returns a VARBIT(1) containing 1.

SELECT CAST( CAST( 8 AS TINYINT ) AS VARBIT ) returns a VARBIT(8) containing 00001000.

SELECT CAST( CAST( 194 AS INTEGER ) AS VARBIT ) returns a VARBIT(32) containing 00000000000000000000000011000010.

Converting binary to bit arrays

When converting a binary type of length \( n \) to a bit array, the length of the array is \( n \times 8 \) bits. The first 8 bits of the bit array become the first byte of the binary value. The most significant bit of the binary value becomes the first bit in the array. The next 8 bits of the bit array become the second byte of the binary value, and so on.

Examples

SELECT CAST( 0x8181 AS VARBIT ) returns a VARBIT(16) containing 1000000110000001.

Converting characters to bit arrays

When converting a character data type of length \( n \) to a bit array, the length of the array is \( n \) bits. Each character must be either '0' or '1' and the corresponding bit of the array is assigned the value 0 or 1.

Example

SELECT CAST( '001100' AS VARBIT ) returns a VARBIT(6) containing 001100.

Converting bit arrays to integers

When converting a bit array to an integer data type, the bit array's binary value is interpreted according to the storage format of the integer type, using the most significant bit first.

Example

SELECT CAST( CAST( '11000010' AS VARBIT ) AS INTEGER ) returns 194 (11000010\(_2\) = 0xC2 = 194).

Converting bit arrays to binary

When converting a bit array to a binary, the first 8 bits of the array become the first byte of the binary value. The first bit of the array becomes the most significant bit of the binary value. The next 8 bits are used as the second byte, and so on. If the length of the bit array is not a multiple of 8, then extra zeros are used to fill the least significant bits of the last byte of the binary value.
Examples

```
SELECT CAST( CAST( '1111' AS VARBIT ) AS BINARY ) returns 0xF0 (1111₂ becomes 11110000₂ = 0xF0).
```

```
SELECT CAST( CAST( '001100000110001' AS VARBIT ) AS BINARY ) returns 0x3031 (001100000110001₂ = 0x3031).
```

Converting bit arrays to characters

When converting a bit array of length \( n \) bits to a character data type, the length of the result is \( n \) characters. Each character in the result is either '0' or '1', corresponding to the bit in the array.

Example

```
SELECT CAST( CAST( '01110' AS VARBIT ) AS VARCHAR ) returns the character string '01110'.
```

Numeric set conversions

When converting a DOUBLE type to a NUMERIC type, precision is maintained for the first 15 significant digits.

See also

- “CAST function [Data type conversion]” on page 174
- “CONVERT function [Data type conversion]” on page 187

Java and SQL data type conversions

Data type conversion between Java types and SQL types is required for both Java stored procedures and JDBC applications. Java to SQL and SQL to Java data type conversions are carried out according to the JDBC standard. The conversions are described in the following tables.

Java to SQL data type conversions

<table>
<thead>
<tr>
<th>Java type</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>CHAR</td>
</tr>
<tr>
<td>String</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>String</td>
<td>TEXT</td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td>MONEY</td>
</tr>
<tr>
<td>Java type</td>
<td>SQL type</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td>SMALLMONEY</td>
</tr>
<tr>
<td>boolean</td>
<td>BIT</td>
</tr>
<tr>
<td>byte</td>
<td>TINYINT</td>
</tr>
<tr>
<td>short</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>int</td>
<td>INTEGER</td>
</tr>
<tr>
<td>long</td>
<td>BIGINT</td>
</tr>
<tr>
<td>float</td>
<td>REAL</td>
</tr>
<tr>
<td>double</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>byte[ ]</td>
<td>VARBINARY</td>
</tr>
<tr>
<td>byte[ ]</td>
<td>IMAGE</td>
</tr>
<tr>
<td>java.sql.Date</td>
<td>DATE</td>
</tr>
<tr>
<td>java.sql.Time</td>
<td>TIME</td>
</tr>
<tr>
<td>java.sql.Timestamp</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>java.lang.Double</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>java.lang.Float</td>
<td>REAL</td>
</tr>
<tr>
<td>java.lang.Integer</td>
<td>INTEGER</td>
</tr>
<tr>
<td>java.lang.Long</td>
<td>BIGINT</td>
</tr>
</tbody>
</table>

## SQL to Java data type conversions

<table>
<thead>
<tr>
<th>SQL type</th>
<th>Java type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>String</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>String</td>
</tr>
<tr>
<td>TEXT</td>
<td>String</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>SQL type</td>
<td>Java type</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>MONEY</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>java.math.BigDecimal (precision=20, scale=0)</td>
</tr>
<tr>
<td>BIT</td>
<td>boolean</td>
</tr>
<tr>
<td>TINYINT</td>
<td>byte</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>int</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>UNSIGNED INTEGER</td>
<td>long</td>
</tr>
<tr>
<td>BIGINT</td>
<td>long</td>
</tr>
<tr>
<td>REAL</td>
<td>float</td>
</tr>
<tr>
<td>FLOAT</td>
<td>double</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>BINARY</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>IMAGE</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>DATE</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>TIME</td>
<td>java.sql.Time</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>java.sql.Timestamp</td>
</tr>
</tbody>
</table>
SQL functions

Functions are used to return information from the database. They can be called anywhere an expression is allowed.

Unless otherwise specified in the documentation, NULL is returned for a function if any argument is NULL.

If an argument is optional, DEFAULT can be provided as an argument.

Functions use the same syntax conventions used by SQL statements.

See also

● “Syntax conventions” on page 422

Function types

This section groups the available function by type.

See also

● “ST_GEOMETRY data type” [UltraLite - Database Management and Reference]

Aggregate functions

Aggregate functions summarize data over a group of rows from the database. The groups are formed using the GROUP BY clause of the SELECT statement. Aggregate functions are allowed only in the SELECT list and in the HAVING and ORDER BY clauses of a SELECT statement.
List of functions

The following aggregate functions are available:

- "ARRAY_AGG function [Aggregate]" on page 159
- "AVG function [Aggregate]"
- "BIT_AND function [Aggregate]"
- "BIT_OR function [Aggregate]"
- "BIT_XOR function [Aggregate]"
- "COVAR_POP function [Aggregate]"
- "COVAR_SAMP function [Aggregate]"
- "COUNT function [Aggregate]"
- "COUNT_BIG function [Aggregate]"
- "CORR function [Aggregate]"
- "FIRST_VALUE function [Aggregate]"
- "GROUPING function [Aggregate]"
- "LAST_VALUE function [Aggregate]"
- "LIST function [Aggregate]"
- "MAX function [Aggregate]"
- "MEDIAN function [Aggregate]"
- "MIN function [Aggregate]"
- "REGR_AVGX function [Aggregate]"
- "REGR_AVGY function [Aggregate]"
- "REGR_COUNT function [Aggregate]"
- "REGR_INTERCEPT function [Aggregate]"
- "REGR_R2 function [Aggregate]"
- "REGR_SLOPE function [Aggregate]"
- "REGR_SXX function [Aggregate]"
- "REGR_SXY function [Aggregate]"
- "REGR_SYY function [Aggregate]"
- "SET_BITS function [Aggregate]"
- "STDDEV function [Aggregate]"
- "STDDEV_POP function [Aggregate]"
- "STDDEV_SAMP function [Aggregate]"
- "SUM function [Aggregate]"
- "VAR_POP function [Aggregate]"
- "VAR_SAMP function [Aggregate]"
- "VARIANCE function [Aggregate]"
- "XMLAGG function [Aggregate]"

Composite functions

Composite functions allow you to perform tasks on arrays.
List of functions

The following composite functions are available:

- “ARRAY constructor [Composite]” on page 157
- “ROW constructor [Composite]” on page 351
- “ARRAY_MAX_CARDINALITY function [Composite]” on page 160
- “CARDINALITY function [Composite]” on page 173
- “TRIM_ARRAY function [Composite]” on page 393

See also

- “ARRAY_AGG function [Aggregate]” on page 159
- “UNNEST array operator”

Bit array functions

Bit array functions allow you to perform tasks on bit arrays.

List of functions

The following bit array functions are available:

- “BIT_AND function [Aggregate]”
- “BIT_OR function [Aggregate]”
- “BIT_XOR function [Aggregate]”
- “BIT_LENGTH function [Bit array]”
- “BIT_SUBSTR function [Bit array]”
- “COUNT_SET_BITS function [Bit array]”
- “GET_BIT function [Bit array]”
- “SET_BIT function [Bit array]”
- “SET_BITS function [Aggregate]”

See also

- “Bitwise operators” on page 19
- “sa_get_bits system procedure” on page 1141

Ranking functions

Ranking functions let you compute a rank value for each row in a result set based on an ordering specified in the query.
List of functions

The following rank functions are available:

- “CUME_DIST function [Ranking]”
- “DENSE_RANK function [Ranking]”
- “PERCENT_RANK function [Ranking]”
- “RANK function [Ranking]”

Data type conversion functions

Data type conversion functions are used to convert arguments from one data type to another, or to test whether they can be converted.

List of functions

The following data type conversion functions are available:

- “BINTOHEX function [Data type conversion]” on page 166
- “CAST function [Data type conversion]”
- “CONVERT function [Data type conversion]”
- “HEXTOBIN function [Data type conversion]” on page 257
- “HEXTOINT function [Data type conversion]”
- “INTTOHEX function [Data type conversion]”
- “ISDATE function [Data type conversion]”
- “ISNUMERIC function [Miscellaneous]”
- “TREAT function [Data type conversion]”

Date and time functions

Date and time functions perform operations on DATE, TIME, TIMESTAMP, and TIMESTAMP WITH TIME ZONE data types.

SQL Anywhere includes compatibility support for Transact-SQL date and time types, including DATETIME and SMALLDATETIME. These Transact-SQL data types are implemented as domains over the native SQL Anywhere TIMESTAMP data type.

Specifying date parts

Many of the date functions use dates built from date parts. The following table displays allowed values of date parts.

When using date and time functions, you can specify a minus sign to subtract from a date or time. For example, to get a timestamp from 31 days ago, you can execute the following:

```
SELECT DATEADD(day, -31, NOW());
```
<table>
<thead>
<tr>
<th>Date part</th>
<th>Abbreviation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>YY</td>
<td>1-9999</td>
</tr>
<tr>
<td>Quarter</td>
<td>QQ</td>
<td>1-4</td>
</tr>
<tr>
<td>Month</td>
<td>MM</td>
<td>1-12</td>
</tr>
<tr>
<td>Week</td>
<td>WK</td>
<td>1-54. Weeks begin on Sunday. A 54-week year occurs in leap years that start on a Saturday.</td>
</tr>
<tr>
<td>Day</td>
<td>DD</td>
<td>1-31</td>
</tr>
<tr>
<td>Dayofyear</td>
<td>DY</td>
<td>1-366</td>
</tr>
<tr>
<td>Weekday</td>
<td>DW</td>
<td>1-7 (Sunday = 1, ..., Saturday = 7)</td>
</tr>
<tr>
<td>Hour</td>
<td>HH</td>
<td>0-23</td>
</tr>
<tr>
<td>Minute</td>
<td>MI</td>
<td>0-59</td>
</tr>
<tr>
<td>Second</td>
<td>SS</td>
<td>0-59</td>
</tr>
<tr>
<td>Millisecond</td>
<td>MS</td>
<td>0-999</td>
</tr>
<tr>
<td>Microsecond</td>
<td>MCS or US</td>
<td>0-999999</td>
</tr>
<tr>
<td>Calyearofweek</td>
<td>CYR</td>
<td>1-9999. The year in which the week begins. The week containing the first few days of the year may have started in the previous year, depending on the weekday on which the year started. Years starting on Monday through Thursday have no days that are part of the previous year, but years starting on Friday through Sunday start their first week on the first Monday of the year.</td>
</tr>
<tr>
<td>Calweekofyear</td>
<td>CWK</td>
<td>1-53. The week number within the year that contains the specified date. For more information about the ISO week system and the ISO 8601 date and time standard, see <a href="http://en.wikipedia.org/wiki/ISO_week_date">http://en.wikipedia.org/wiki/ISO_week_date</a>.</td>
</tr>
<tr>
<td>Caldayofweek</td>
<td>CDW</td>
<td>1-7. (Monday = 1, ..., Sunday = 7)</td>
</tr>
<tr>
<td>TZOOffset</td>
<td>TZ</td>
<td>-840 to 840</td>
</tr>
</tbody>
</table>
List of functions

The following date and time functions are available:

- “DATE function [Date and time]”
- “DATEADD function [Date and time]”
- “DATEDIFF function [Date and time]”
- “DATEFORMAT function [Date and time]”
- “DATENAME function [Date and time]”
- “DATEPART function [Date and time]”
- “DATEDATE function [Date and time]”
- “DAY function [Date and time]”
- “DAYNAME function [Date and time]”
- “DAYS function [Date and time]”
- “DOW function [Date and time]”
- “GETDATE function [Date and time]”
- “HOUR function [Date and time]”
- “HOURS function [Date and time]”
- “MINUTE function [Date and time]”
- “MINUTES function [Date and time]”
- “MONTH function [Date and time]”
- “MONTHNAME function [Date and time]”
- “MONTHS function [Date and time]”
- “NOW function [Date and time]”
- “QUARTER function [Date and time]”
- “SECOND function [Date and time]”
- “SECONDS function [Date and time]”
- “SWITCHOFFSET function [Date and time]”
- “SYSDATETIMEOFFSET function [Date and time]”
- “TODAY function [Date and time]”
- “TODATETIMEOFFSET function [Date and time]”
- “WEEKS function [Date and time]”
- “YEAR function [Date and time]”
- “YEARS function [Date and time]”
- “YMD function [Date and time]”

See also

- “Date and time data types” on page 108
- “UltraLite SQL data types” [UltraLite - Database Management and Reference]

User-defined functions

A user-defined function, or UDF, is a function created by the user of a program or environment. User-defined functions are in contrast to functions that are built in to the program or environment.

There are two mechanisms for creating user-defined functions in SQL Anywhere. You can use SQL or any CLR language to write the function.
User-defined functions in SQL

You can implement your own functions in SQL by using the CREATE FUNCTION statement.

The RETURN statement inside the CREATE FUNCTION statement determines the data type of the function.

Once a SQL user-defined function is created, it can be used anywhere a built-in function of the same data type is used.

User-defined functions in Java and the CLR

Java classes provide a more powerful and flexible way of implementing user-defined functions, with the additional advantage that they can be moved from the database server to a client application if desired. Any class method of an installed Java class can be used as a user-defined function anywhere a built-in function of the same data type is used. Instance methods are tied to particular instances of a class, and so have different behavior from standard user-defined functions.

SQL Anywhere includes support for CLR stored procedures and functions. A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a .NET language such as C# or Visual Basic, and the execution of the procedure or function takes place outside the database server (that is, within a separate .NET executable). Only .NET version 2.0 is supported.

Deciding whether to create a user-defined function or procedure

Functions are similar to procedures. Deciding whether to create a function or a procedure depends on what you want returned, and the object will be called. When deciding whether to create a UDF or a procedure, consider their unique characteristics listed below.

Functions:

- can return a single value of arbitrary type, and allow you to declare the returned type using the RETURNS clause
- can be used in most places an expression can be used
- allow you to define only IN parameters

Procedures:

- can return multiple values using INOUT or OUT parameters
- can return result sets
- can be referenced in the FROM clause of a query, or using a CALL statement, or using a Transact-SQL EXECUTE statement
- can be called using named parameters
Miscellaneous functions

Miscellaneous functions perform operations on arithmetic, string, or date/time expressions, including the return values of other functions.

List of functions

The following miscellaneous functions are available:

- “ARGIN function [Miscellaneous]”
- “COALESCE function [Miscellaneous]”
- “CONFLICT function [Miscellaneous]”
- “ERRORMSG function [Miscellaneous]”
- “ESTIMATE function [Miscellaneous]”
- “ESTIMATE_SOURCE function [Miscellaneous]”
- “EXPERIENCE_ESTIMATE function [Miscellaneous]”
- “EXPLANATION function [Miscellaneous]”
- “EXPRTYPE function [Miscellaneous]”
- “GET_IDENTITY function [Miscellaneous]”
- “GRAPHICAL_PLAN function [Miscellaneous]”
- “GREATER function [Miscellaneous]”
- “IDENTITY function [Miscellaneous]”
- “IFNULL function [Miscellaneous]”
- “INDEX_ESTIMATE function [Miscellaneous]”
- “ISNULL function [Miscellaneous]”
- “LESSER function [Miscellaneous]”
- “NEWID function [Miscellaneous]”
- “NULLIF function [Miscellaneous]”
- “NUMBER function [Miscellaneous]”
- “PLAN function [Miscellaneous]”
- “REWRITE function [Miscellaneous]”
- “ROW_NUMBER function [Miscellaneous]”
- “SQLDIALECT function [Miscellaneous]”
- “SQLFLAGGER function [Miscellaneous]”
- “ERROR_LINE function [Miscellaneous]” on page 228
- “TRACEBACK function [Miscellaneous]”
- “TRANSACTSQL function [Miscellaneous]”
- “VAREXISTS function [Miscellaneous]”
- “WATCOMSQL function [Miscellaneous]”
Numeric functions

Numeric functions perform mathematical operations on numerical data types or return numeric information.

List of functions

The following numeric functions are available:

- “ABS function [Numeric]”
- “ACOS function [Numeric]”
- “ASIN function [Numeric]”
- “ATAN function [Numeric]”
- “ATAN2 function [Numeric]”
- “CEILING function [Numeric]”
- “COS function [Numeric]”
- “COT function [Numeric]”
- “DEGREES function [Numeric]”
- “EXP function [Numeric]”
- “FLOOR function [Numeric]”
- “LOG function [Numeric]”
- “LOG10 function [Numeric]”
- “MOD function [Numeric]”
- “PI function [Numeric]”
- “POWER function [Numeric]”
- “RADIANS function [Numeric]”
- “RAND function [Numeric]”
- “REMAINDER function [Numeric]”
- “ROUND function [Numeric]”
- “SIGN function [Numeric]”
- “SIN function [Numeric]”
- “SQRT function [Numeric]”
- “TAN function [Numeric]”
- “TRUNCNUM function [Numeric]”

Web services functions

HTTP functions assist the handling of HTTP requests within web services. Likewise, SOAP functions assist the handling of SOAP requests within web services.
The following functions are available:

- “HTML_DECODE function [Miscellaneous]”
- “HTML_ENCODE function [Miscellaneous]”
- “HTTP_BODY function [Web service]”
- “HTTP_DECODER function [Web service]”
- “HTTP_ENCODE function [Web service]”
- “HTTP_HEADER function [Web service]”
- “HTTP_RESPONSE_HEADER function [Web service]”
- “HTTP_VARIABLE function [Web service]”
- “NEXT_HTTP_HEADER function [Web service]”
- “NEXT_HTTP_RESPONSE_HEADER function [Web service]”
- “NEXT_HTTP_VARIABLE function [Web service]”
- “NEXT_SOAP_HEADER function [SOAP]”
- “SOAP_HEADER function [SOAP]”

There are also system procedures available for web services.

See also

- “SQL Anywhere as an HTTP web server” [SQL Anywhere Server - Programming]
- “-xs database server option” [SQL Anywhere Server - Database Administration]
- “Web services system procedures” on page 1085

String functions

String functions perform conversion, extraction, or manipulation operations on strings, or return information about strings.

When working in a multibyte character set, check carefully whether the function being used returns information concerning characters or bytes.
List of functions

The following string functions are available:

- “ASCII function [String]”
- “BASE64_DECODE function [String]”
- “BASE64_ENCODE function [String]”
- “BYTE_LENGTH function [String]”
- “BYTE_SUBSTR function [String]”
- “CHAR function [String]”
- “CHARINDEX function [String]”
- “CHAR_LENGTH function [String]”
- “COMPARE function [String]”
- “COMPRESS function [String]”
- “CSCONVERT function [String]”
- “DECOMPRESS function [String]”
- “DECRYPT function [String]”
- “DIFFERENCE function [String]”
- “ENCRYPT function [String]”
- “HASH function [String]”
- “INSERTSTR function [String]”
- “LCASE function [String]”
- “LEFT function [String]”
- “LENGTH function [String]”
- “LOCATE function [String]”
- “LOWER function [String]”
- “LTRIM function [String]”
- “NCHAR function [String]”
- “PATINDEX function [String]”
- “READ_CLIENT_FILE function [String]”
- “REGEXP_SUBSTR function [String]”
- “REPEAT function [String]”
- “REPLACE function [String]”
- “REPLICATE function [String]”
- “REVERSE function [String]”
- “RIGHT function [String]”
- “RTRIM function [String]”
- “SIMILAR function [String]”
- “SORTKEY function [String]”
- “SOUNDEX function [String]”
- “SPACE function [String]”
- “STR function [String]”
- “STRING function [String]”
- “STRTOUUID function [String]”
- “STUFF function [String]”
- “SUBSTRING function [String]”
- “TO_CHAR function [String]”
- “TO_NCHAR function [String]”
System functions

System functions return system information.

List of functions

The following system functions are available:

- “CONNECTION_EXTENDED_PROPERTY function [String]”
- “CONNECTION_PROPERTY function [System]”
- “DATALENGTH function [System]”
- “DB_ID function [System]”
- “DB_NAME function [System]”
- “DB_EXTENDED_PROPERTY function [System]”
- “DB_PROPERTY function [System]”
- “EVENT_CONDITION function [System]”
- “EVENT_CONDITION_NAME function [System]”
- “EVENT_PARAMETER function [System]”
- “NEXT_CONNECTION function [System]”
- “NEXT_DATABASE function [System]”
- “PROPERTY function [System]”
- “PROPERTY_DESCRIPTION function [System]”
- “PROPERTY_NAME function [System]”
- “PROPERTY_NUMBER function [System]”
- “SUSER_ID function [System]”
- “SUSER_NAME function [System]”
- “TSEQUAL function [System] (deprecated)”
- “USER_ID function [System]”
- “USER_NAME function [System]”

Notes

- The db_id, db_name, and datalength functions are implemented as built-in functions.

- Some system functions are implemented in SQL Anywhere as stored procedures.

System functions that are not described elsewhere are noted in the following table.
<table>
<thead>
<tr>
<th>System function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL_LENGTH( @object_name, @column_name )</td>
<td>Returns the INTEGER defined length of the specified column</td>
</tr>
<tr>
<td>COL_NAME( @object_id, @column_id [, @database_id ] )</td>
<td>Returns the CHAR(128) column name</td>
</tr>
<tr>
<td>INDEX_COL( @table_name, @index_id, @key_# [, @user_id ] )</td>
<td>Returns the CHAR(128) name of the indexed column</td>
</tr>
<tr>
<td>OBJECT_ID( @object_name )</td>
<td>Returns the INTEGER object ID</td>
</tr>
<tr>
<td>OBJECT_NAME( @object_id [, @database_id ] )</td>
<td>Returns the CHAR(128) object name</td>
</tr>
</tbody>
</table>

### Text and image functions

Text and image functions operate on text and image data types. SQL Anywhere supports only the `textptr` text and image function.

### List of functions

The following text and image function is available:

- “TEXTPTR function [Text and image]”

### Functions

Each function is listed, and the function type (numeric, character, and so on) is indicated next to it.

For links to all functions of a given type, see “Function types” on page 143.

### ABS function [Numeric]

Returns the absolute value of a numeric expression.

**Syntax**

ABS( numeric-expression )

**Parameters**

- **numeric-expression** The number whose absolute value is to be returned.

**Returns**

An absolute value of the numeric expression.
<table>
<thead>
<tr>
<th>Numeric-expression data type</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>INT</td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>NUMERIC</td>
</tr>
</tbody>
</table>

**Standards and compatibility**
- SQL/2008 The ABS function is part of optional SQL/2008 language feature T441.

**Example**
The following statement returns the value 66:

```
SELECT ABS( -66 );
```

**ACOS function [Numeric]**

Returns the arc-cosine, in radians, of a numeric expression.

**Syntax**

```
ACOS( numeric-expression )
```

**Parameters**
- `numeric-expression` The cosine of the angle.

**Returns**

DOUBLE

**Remarks**
This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

**See also**
- “ASIN function [Numeric]” on page 161
- “ATAN function [Numeric]” on page 162
- “ATAN2 function [Numeric]” on page 163
- “COS function [Numeric]” on page 191

**Standards and compatibility**
- SQL/2008 Vendor extension.
Example

The following statement returns the arc-cosine value for 0.52:

```
SELECT ACOS( 0.52 );
```

**ARGN function [Miscellaneous]**

Returns a selected argument from a list of arguments.

**Syntax**

```
ARGN( integer-expression, expression [, ... ] )
```

**Parameters**

- `integer-expression`  The position of an argument within the list of expressions.
- `expression`  An expression of any data type passed into the function. All supplied expressions must be of the same data type.

**Returns**

Using the value of the `integer-expression` as n, returns the nth argument (starting at 1) from the remaining list of arguments.

**Remarks**

While the expressions can be of any data type, they must all be of the same data type. The integer expression must be from one to the number of expressions in the list or NULL is returned. Multiple expressions are separated by a comma.

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

The following statement returns the value 6:

```
SELECT ARGN( 6, 1,2,3,4,5,6 );
```

**ARRAY constructor [Composite]**

Returns elements of a specific data type.

**Syntax 1**

```
ARRAY( expression [, expression ... ] )
```

**Syntax 2**

```
ARRAY( single-column-query-expression )
```
Parameters

- **expression**  An element expression in the ROW type.
- **single-column-query-expression**  A query statement that returns a single column.

Returns

Array value

Remarks

All expressions must be union compatible.

All elements are initialized to NULL, and remain NULL until a value is explicitly or implicitly placed within a particular array element.

An ARRAY type can contain other ARRAY or ROW values or be part of a ROW type.

The FETCH statement supports the transfer of values into an array. You can fetch values into an array for individual expressions, for entire arrays, or for portions of arrays.

Specific values or vectors of values can be dereferenced by using double square brackets.

See also

- “Composite data types” on page 128
- “Composite functions” on page 144
- “Comparisons of composite types” on page 136
- “FETCH statement [ESQL] [SP]” on page 800

Standards and compatibility

- **SQL/2008**  Vendor extension.
- **Oracle**  VARRAY can be used as a synonym of ARRAY.

Example

The following example illustrates how to construct an array:

```sql
SELECT FIRST f[2] FROM ( SELECT ARRAY( ID,Quantity ) FROM GROUPO.Products )
AS dt( f ) ORDER BY f[1] ASC;
```

In this example, for each row of the Products table the ARRAY constructor builds an ARRAY type with two elements: the column ID and the value of the Quantity column, both of which are integers. The result is ordered by the first element of the array in each row, and the result returned is the second element from the array with the smallest first element (the Product ID of 300). You can also construct an array directly from a single-column query expression.

The following example illustrates an alternative way of constructing an array:

```sql
SELECT * FROM GROUPO.SalesOrders S WHERE ARRAY( SELECT P.ID FROM GROUPO.Products P JOIN GROUPO.SalesOrderItems SI ON( P.ID = SI.ProductID )AND SI.ID = S.ID ORDER BY P.ID )< ARRAY ( SELECT ID FROM GROUPO.Products ORDER BY ID );
```
In the following example, the query's SELECT list uses three arrays: one produces a GROUP BY expression, and the MAX function uses the others. Each ARRAY type is de-referenced for a specific element before the result is returned to the client:

```sql
SELECT FIRST ARRAY( Quantity )[[1]], MAX( ARRAY( ID,Quantity ) )[[1]],
       MAX( ARRAY( name,name ))[[2]] FROM Products GROUP BY ARRAY( Quantity )
       ORDER BY 1;
```

The following example illustrates how to use the FETCH statement to transfer values into an array:

```sql
BEGIN
    DECLARE product_orders ARRAY(10) OF ARRAY OF INTEGER;
    DECLARE products ARRAY(10) OF INTEGER;
    DECLARE greatest_orders INTEGER = 0;
    DECLARE i INTEGER = 1;
    DECLARE curs CURSOR FOR
        SELECT ProductID,
            ARRAY_AGG( Quantity ) AS Quantities
        FROM GROUPO.SalesOrderItems
        GROUP BY ProductID
        ORDER BY ProductID;

    OPEN curs;
    lp: LOOP
        FETCH NEXT curs INTO products[[i]], product_orders[[i]];
        IF SQLCODE <> 0 THEN LEAVE lp; END IF;
        IF i = 1 THEN
            SET greatest_orders = 1;
        ELSE
            IF CARDINALITY( product_orders[[greatest_orders]] )
                < CARDINALITY( product_orders[[i]] ) THEN
                SET greatest_orders = i;
            END IF;
        END IF;
        SET i = i + 1;
    END LOOP;

    IF greatest_orders >= 1 THEN
        SELECT * FROM GROUPO.Products WHERE ID = products[[greatest_orders]];
    END IF;
END;
```

**ARRAY_AGG function [Aggregate]**

Creates an unbounded, single-dimensional array from the specified expression for each group where the array element type is identical to the specified expression.

**Syntax**

```sql
ARRAY_AGG( expression
             [ ORDER BY order-by-expression [ ASC | DESC ], ... ]
)
```

**Parameters**

- **expression**  The expression to base the array on.

  The array is created with the first element having the value of the first group from *expression*, the second element having the value of the second group, and so on.
• **s**  A set of attributes that indicate how sort the rows returned by the *expression*.

**Returns**

ARRAY

**Remarks**

Array elements are filled from the input beginning with the first element.

ARRAY\_AGG does not ignore NULL values in its input. NULL values are stored in the array as separate elements like any other value. If the group is empty, the result of the ARRAY\_AGG function contains a NULL element for that group.

ARRAY\_AGG cannot be used as a window function, but it can be used as an input to a window function.

The UNNEST array operator can be used to create a series of rows from an array to process each array element with other relational expressions.

**See also**

- “UNNEST array operator” on page 18
- “LIST function [Aggregate]” on page 285

**Standards and compatibility**

- SQL/2008  Vendor extension.

---

**ARRAY\_MAX\_CARDINALITY function [Composite]**

Returns the maximal number of elements in the array.

**Syntax**

\[
\text{ARRAY\_MAX\_CARDINALITY( array-expression )}
\]

**Parameters**

- **array-expression**  The array expression to evaluate.

If *array-expression* is NULL, then ARRAY\_MAX\_CARDINALITY returns NULL.

**Returns**

INTEGER

**Remarks**

The cardinality of the collection is the number of elements in the collection.

For an unbounded array, ARRAY\_MAX\_CARDINALITY returns the maximum size limit of an array supported by SQL Anywhere. For a bounded array or an array composed via a constructor, ARRAY\_MAX\_CARDINALITY returns the maximum explicitly or implicitly declared size of the array.
See also

- “ARRAY_AGG function [Aggregate]” on page 159
- “CARDINALITY function [Composite]” on page 173
- “TRIM_ARRAY function [Composite]” on page 393

Standards and compatibility

- SQL/2008  Vendor extension.

**ASCII function [String]**

Returns the integer ASCII value of the first byte in a string-expression.

**Syntax**

```sql
ASCII( string-expression )
```

**Parameters**

- `string-expression`  The string.

**Returns**

SMALLINT

**Remarks**

If the string is empty, then ASCII returns zero. Literal strings must be enclosed in quotes. If the database character set is multibyte and the first character of the parameter string consists of more than one byte, the result is NULL.

See also

- “CHAR function [String]” on page 176
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

**Example**

The following statement returns the value 90:

```sql
SELECT ASCII( 'Z' );
```

**ASIN function [Numeric]**

Returns the arc-sine, in radians, of a number.

**Syntax**

```sql
ASIN( numeric-expression )
```
Parameters

- `numeric-expression` The sine of the angle.

Returns

DOUBLE

Remarks

The SIN and ASIN functions are inverse operations.

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

See also

- “ACOS function [Numeric]” on page 156
- “ATAN function [Numeric]” on page 162
- “ATAN2 function [Numeric]” on page 163
- “SIN function [Numeric]” on page 362

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns the arc-sine value for 0.52:

```
SELECT ASIN( 0.52 );
```

## ATAN function [Numeric]

Returns the arc-tangent, in radians, of a number.

Syntax

```
ATAN( numeric-expression )
```

Parameters

- `numeric-expression` The tangent of the angle.

Remarks

The ATAN and TAN functions are inverse operations.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.
ATAN2 function [Numeric]

Returns the arc-tangent, in radians, of the ratio of two numbers.

Syntax

\{ ATN2 | ATAN2 \}( numeric-expression-1, numeric-expression-2 )

Parameters

- **numeric-expression-1**  The numerator in the ratio whose arc-tangent is calculated.
- **numeric-expression-2**  The denominator in the ratio whose arc-tangent is calculated.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

See also

- “ACOS function [Numeric]” on page 156
- “ASIN function [Numeric]” on page 161
- “ATAN function [Numeric]” on page 162
- “TAN function [Numeric]” on page 385

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the arc-tangent value for the ratio 0.52 to 0.60:

```
SELECT ATAN2( 0.52, 0.60 );
```
AVG function [Aggregate]

Computes the average, for a set of rows, of a numeric expression or of a set of unique values.

Syntax 1

\[
\text{AVG}( \ [ \text{ALL} \ | \text{DISTINCT} \ ] \ \text{numeric-expression} )
\]

Syntax 2

\[
\text{AVG}( \ [ \text{ALL} \ ] \ \text{numeric-expression} ) \ \text{OVER} ( \ \text{window-spec} )
\]

\[
\text{window-spec} : \text{see Syntax 2 instructions in the Remarks section below}
\]

Parameters

- \([\text{ALL} ]\ \text{numeric-expression}\) The expression whose average is calculated over the rows in each group.
- \text{DISTINCT clause} Computes the average of the unique numeric values in each group.

Returns

Returns the NULL value for a group containing no rows.

Returns DOUBLE if the argument is DOUBLE, otherwise NUMERIC.

Remarks

This average does not include rows where the \text{numeric-expression} is the NULL value.

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of \text{window-spec} can be specified either in the function syntax (inline), or with a \text{WINDOW} clause in the SELECT statement. See the \text{window-spec} definition in “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an \text{OVER} clause, see “Window definition: Inlining using the \text{OVER} clause and \text{WINDOW} clause” [SQL Anywhere Server - SQL Usage].

This function can generate an overflow error, resulting in an error being returned. You can use the CAST function on \text{numeric-expression} to avoid the overflow error.

See also

- “CAST function [Data type conversion]” on page 174
- “SUM function [Aggregate]” on page 381
- “COUNT function [Aggregate]” on page 192

Standards and compatibility

- \text{SQL/2008} AVG is a core feature of the SQL/2008 standard.
- \text{SQL/2008} Syntax 1 is a core feature of the SQL/2008 standard, while Syntax 2 comprises part of optional SQL/2008 language feature T611, "Basic OLAP operations". The ability to specify
DISTINCT over an expression that is not a column reference comprises part of optional SQL language feature F561, "Full value expressions". SQL Anywhere also supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the AVG function, combined with an outer reference.

See also

- “Troubleshooting: Aggregate functions and outer references” [SQL Anywhere 16 - Changes and Upgrading]

Example

The following statement returns the value 49988.623200 when connected to the SQL Anywhere 16 Demo:

```
SELECT AVG( Salary ) FROM Employees;
```

The following statement returns the average product price from the Products table when connected to the SQL Anywhere 16 Demo:

```
SELECT AVG( DISTINCT UnitPrice ) FROM Products;
```

The following statement returns an error with SQLSTATE 42W68 because the arguments of AVG contain both a quantified expression from the subquery, and an outer reference (p.Quantity) from the outer SELECT block:

```
SELECT * from GROUPO.Products as p
WHERE p.Quantity > ( SELECT AVG( 0.5 * p.Quantity + 0.5 * s.Quantity )
from GROUPO.SalesOrderItems as s
WHERE s.ProductID = p.ProductID )
```

### BASE64_DECODE function [String]

Decodes data using the MIME base64 format and returns the string as a LONG VARCHAR.

Syntax

```
BASE64_DECODE( string-expression )
```

Parameters

- **string-expression** The string that is to be decoded. The string must be base64-encoded.

Returns

LONG VARCHAR
See also

- “BASE64_ENCODE function [String]” on page 166
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following inserts an image into an image table from an embedded SQL program. The input data (host variable) must be base64 encoded:

```
EXEC SQL INSERT INTO images (image_data) VALUES (BASE64_DECODE(:img));
```

### BASE64_ENCODE function [String]

Encodes data using the MIME base64 format and returns it as a 7-bit ASCII string.

**Syntax**

```
BASE64_ENCODE(string-expression)
```

**Parameters**

- `string-expression` The string that is to be encoded.

**Returns**

LONG VARCHAR

See also

- “BASE64_DECODE function [String]” on page 165
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following retrieves data from a fictitious table containing images and returns it in ASCII format. The resulting string can be embedded into an email message, and then decoded by the recipient to retrieve the original image.

```
SELECT BASE64_ENCODE(image_data) FROM IMAGES;
```

### BINTOHEX function [Data type conversion]

Returns the hexadecimal equivalent of a binary string.
Syntax

\texttt{BINTOHEX( binary-expression )}

Parameters

- \textit{binary-expression}  The binary string to be converted to a hexadecimal string.

Returns

The BINTOHEX function returns a LONG VARCHAR string. The length of the result is twice the length of the input string.

Remarks

The CAST, CONVERT, BINTOHEX, HEXTOBIN, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values. For more information about using these functions, see “Converting to and from hexadecimal values” on page 7.

See also

- “HEXTOBIN function [Data type conversion]” on page 257

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns a string containing 313233:

\begin{verbatim}
SELECT BINTOHEX(0x313233);
\end{verbatim}

\section*{BIT_AND function [Aggregate]}

Returns the bit-wise AND of the specified expression for each group of rows.

Syntax

\texttt{BIT_AND( bit-expression )}

Parameters

- \textit{bit-expression}  The object to be aggregated. The expression can be a VARBIT array, a BINARY value, or an INTEGER (including all integer variants such as BIT and TINYINT).

Returns

The same data type as the argument. For each bit position compared, if every row has a 1 in the bit position, return 1; otherwise, return 0.

See also

- “BIT_OR function [Aggregate]” on page 169
- “BIT_XOR function [Aggregate]” on page 171
- “Bitwise operators” on page 19
Standards and compatibility

- SQL/2008    Vendor extension.

Example

The following example generates four rows containing a CHAR column, then converts the values to VARBIT:

```sql
SELECT BIT_AND( CAST(row_value AS VARBIT) )
FROM dbo.sa_split_list('0001,0111,0100,0011');
```

The result 0000 is determined as follows:

1. A bitwise AND is performed between row 1 (0001) and row 2 (0111), resulting in 0001 (both values had a 1 in the fourth bit).
2. A bitwise AND is performed between the result from the previous comparison (0001) and row 3 (0100), resulting in 0000 (neither value had a 1 in the same bit).
3. A bitwise AND is performed between the result from the previous comparison (0000) and row 4 (0011), resulting in 0000 (neither value had a 1 in the same bit).

**BIT_LENGTH function [Bit array]**

Returns the number of bits stored in the array.

Syntax

```
BIT_LENGTH( bit-expression )
```

Parameters

- **bit-expression**    The bit expression for which the length is to be determined.

Returns

INT

See also

- “CHAR_LENGTH function [String]” on page 177

Standards and compatibility

- SQL/2008    Vendor extension.

- SQL/1999    The BIT_LENGTH function was a core feature of the SQL/1999 standard. The BIT VARYING data type was optional language feature F511 of the SQL/1999 standard. Support for BIT_LENGTH and the BIT VARYING data type were removed in the SQL/2003 standard.

Example

The following statement returns the value 8:

```sql
SELECT BIT_LENGTH( '01101011' );
```
**BIT_OR function [ Aggregate ]**

Returns the bit-wise OR of the specified expression for each group of rows.

**Syntax**

\[
\text{BIT\_OR( } \text{bit-expression } \text{)}
\]

**Parameters**

- **bit-expression**  
The object to be aggregated. The expression can be a VARBIT array, a BINARY value, or an INTEGER (including all integer variants such as BIT and TINYINT).

**Returns**

The same data type as the argument. For each bit position compared, if any row has a 1 in the bit position, this function returns 1; otherwise, it returns 0.

**See also**

- “BIT_AND function [ Aggregate ]” on page 167
- “BIT_XOR function [ Aggregate ]” on page 171
- “Bitwise operators” on page 19

**Standards and compatibility**

- SQL/2008  
  Vendor extension.

**Example**

The following example generates four rows containing a CHAR column, then converts the values to VARBIT:

\[
\text{SELECT BIT\_OR( CAST(row_value AS VARBIT) )}
\]

\[
\text{FROM dbo.sa_split_list('0001,0111,0100,0011');}
\]

The result 0111 is determined as follows:

1. A bitwise OR is performed between row 1 (0001) and row 2 (0111), resulting in 0111.
2. A bitwise OR is performed between the result from the previous comparison (0111) and row 3 (0100), resulting in 0111.
3. A bitwise OR is performed between the result from the previous comparison (0111) and row 4 (0011), resulting in 0111.

**BIT_SUBSTR function [ Bit array ]**

Returns a sub-array of a bit array.

**Syntax**

\[
\text{BIT\_SUBSTR( } \text{bit-expression } [ , \text{ start } [ , \text{ length } ] ] )
\]
Parameters

- **bit-expression**  The bit array from which the sub-array is to be extracted.

- **start**  The start position of the sub-array to return. A negative starting position specifies the number of bits from the end of the array instead of the beginning. The first bit in the array is at position 1.

- **length**  The length of the sub-array to return. A positive length specifies that the sub-array ends length bits to the right of the starting position, while a negative length returns, at most, length bits up to, and including, the starting position, from the left of the starting position.

Returns

LONG VARBIT

Remarks

Both start and length can be either positive or negative. Using appropriate combinations of negative and positive numbers, you can get a sub-array from either the beginning or end of the string. Using a negative number for length does not impact the order of the bits returned in the sub-array.

If length is specified, the sub-array is restricted to that length. If start is zero and length is non-negative, a start value of 1 is used. If start is zero and length is negative, a start value of -1 is used.

If length is not specified, selection continues to the end of the array.

The BIT_SUBSTR function is equivalent to, but faster than, the following:

```sql
CAST( SUBSTR( CAST( bit-expression AS VARCHAR ),
start [, length ] )
AS VARBIT );
```

See also

- “SUBSTRING function [String]” on page 379

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statement returns 1101:

```sql
SELECT BIT_SUBSTR( '001101', 3 );
```

The following statement returns 10110:

```sql
SELECT BIT_SUBSTR( '01011011101111011111', 2, 5 );
```

The following statement returns 11111:

```sql
SELECT BIT_SUBSTR( '01011011101111011111', -5, 5 );
```
**BIT_XOR function [Aggregate]**

Returns the bit-wise XOR of the specified expression for each group of rows.

**Syntax**

```
BIT_XOR( bit-expression )
```

**Parameters**

- `bit-expression` The object to be aggregated. The expression can be a VARBIT array, a BINARY value, or an INTEGER (including all integer variants such as BIT and TINYINT).

**Returns**

The same data type as the argument. For each bit position compared, if an odd number of rows have a 1 in the bit position, return 1; otherwise, return 0.

**See also**

- “BIT_AND function [Aggregate]” on page 167
- “BIT_OR function [Aggregate]” on page 169
- “Bitwise operators” on page 19

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following example generates four rows containing a CHAR column, then converts the values to VARBIT:

```
SELECT BIT_XOR( CAST(row_value AS VARBIT) )
FROM dbo.sa_split_list('0001,0111,0100,0011');
```

The result 0001 is determined as follows:

1. A bitwise exclusive OR (XOR) is performed between row 1 (0001) and row 2 (0111), resulting in 0110.
2. A bitwise XOR is performed between the result from the previous comparison (0110) and row 3 (0100), resulting in 0010.
3. A bitwise XOR is performed between the result from the previous comparison (0010) and row 4 (0011), resulting in 0001.

**BYTE_LENGTH function [String]**

Returns the number of bytes in a string.

**Syntax**

```
BYTE_LENGTH( string-expression )
```
Parameters
● string-expression The string whose length is to be calculated.

Returns
INT

Remarks
Trailing white space characters in the string-expression are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the BYTE_LENGTH value may differ from the number of characters returned by CHAR_LENGTH.

This function supports NCHAR inputs and/or outputs.

See also
● “CHAR_LENGTH function [String]” on page 177
● “DATALENGTH function [System]” on page 201
● “LENGTH function [String]” on page 283
● “String functions” on page 152

Standards and compatibility
● SQL/2008 Vendor extension. The equivalent function in the SQL/2008 standard is the OCTET_LENGTH function.

Example
The following statement returns the value 12:

SELECT BYTE_LENGTH( 'Test Message' );

BYTE_SUBSTR function [String]

Returns a substring of a string. The substring is calculated using bytes, not characters.

Syntax
BYTE_SUBSTR( string-expression, start [, length ] )

Parameters
● string-expression The string from which the substring is taken.

● start An integer expression indicating the start of the substring. A positive integer starts from the beginning of the string, with the first character being position 1. A negative integer specifies a substring starting from the end of the string, the final character being at position -1.
• **length**  An integer expression indicating the length of the substring. A positive *length* specifies the number of bytes to be taken *starting* at the start position. A negative *length* returns at most *length* bytes up to, and including, the starting position.

**Returns**

BINARY, VARCHAR, or NVARCHAR. The value returned depends on the type of *string-expression*. Also, the arguments you specify determine if the returned value is LONG. For example, LONG is not returned when you specify a constant < 32K for *length*.

**Remarks**

If *length* is specified, the substring is restricted to that number of bytes. Both *start* and *length* can be either positive or negative. Using appropriate combinations of negative and positive numbers, you can get a substring from either the beginning or end of the string.

If *start* is zero and *length* is non-negative, a *start* value of 1 is used. If *start* is zero and *length* is negative, a start value of -1 is used.

**See also**

- “SUBSTRING function [String]” on page 379
- “String functions” on page 152

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

The following statement returns the value Test:

```
SELECT BYTE_SUBSTR( 'Test Message', 1, 4 );
```

**CARDINALITY function [Composite]**

Returns the highest number of any array element that has been assigned a value, including NULL.

**Syntax**

```
CARDINALITY( array-expression )
```

**Parameters**

- **array-expression**  The array expression on which the cardinality is calculated.

  If *array-expression* is NULL, then CARDINALITY returns NULL.

**Returns**

INTEGER

**Remarks**

The cardinality of the collection is the number of elements in the collection.
The result is an integer between zero and the maximum size of the array.

See also
- “ARRAY_AGG function [Aggregate]” on page 159
- “ARRAY_MAX_CARDINALITY function [Composite]” on page 160
- “TRIM_ARRAY function [Composite]” on page 393

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example returns the value 4:

```
SELECT CARDINALITY( ARRAY( 3,4 ) || ARRAY( 5,6 ) );
```

CAST function [Data type conversion]
Returns the value of an expression converted to a supplied data type.

Syntax
```
CAST( expression AS datatype )
```

Parameters
- `expression` The expression to be converted.
- `datatype` The target data type.

Returns
Depends on the data type requested.

Remarks
If you do not indicate a length for character string types, an appropriate length is chosen. If neither precision nor scale is specified for a DECIMAL conversion, the database server selects appropriate values.

If you use the CAST function to truncate strings, the string_rtruncation database option must be set to OFF; otherwise, there will be an error. It is recommended that you use the LEFT function to truncate strings.

The HEXTOINT and INTTOHEX functions can be used to convert to and from hexadecimal values. For more information about using these functions, see “Converting to and from hexadecimal values” on page 7.
Standards and compatibility

- **SQL/2008**  The CAST function is a core feature of the SQL/2008 standard. However, in SQL Anywhere CAST supports a number of data type conversions that are not permitted by the SQL standard. For example, in SQL Anywhere you can CAST an integer value to a DATE type, whereas in the SQL standard this type conversion is not permitted. For more information, see “Data type conversions” on page 137.

Example

The following function ensures a string is used as a date:

```
SELECT CAST( '2000-10-31' AS DATE );
```

The value of the expression `1 + 2` is calculated, and the result is then cast into a single-character string.

```
SELECT CAST( 1 + 2 AS CHAR );
```

Casting between VARCHAR and ST_GEOMETRY is usually implicit. For example, the following statement adds values to ST_GEOMETRY columns using the ST_POINT function and a VARCHAR. Each value is implicitly cast to an ST_GEOMETRY data type consistent with the table columns, but results still appear as VARCHAR.

```
INSERT INTO T1 VALUES (2, ST_POINT(1,2,0), 'SRID=2163;Point(1 2)');
```

### CEILING function [Numeric]

Returns the first integer that is greater or equal to a given value. For positive numbers, this is known as rounding up.

**Syntax**

```
{ CEILING | CEIL } ( numeric-expression )
```

**Parameters**

- **numeric-expression**  The number whose ceiling is to be calculated.

**Returns**

DOUBLE

**Remarks**

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.
CHAR function [String]

Returns the character with the ASCII value of a number.

Syntax

CHAR( integer-expression )

Parameters

- integer-expression The number to be converted to an ASCII character. The number must be in the range 0 to 255, inclusive.

Returns

VARCHAR

Remarks

The character returned corresponds to the supplied numeric expression in the current database character set, according to a binary sort order.

CHAR returns NULL for integer expressions with values greater than 255 or less than zero.

See also

- “String functions” on page 152

Standards and compatibility

- SQL/2008 Vendor extension.
CHAR_LENGTH function [String]

Returns the number of characters in a string.

Syntax

```
CHAR_LENGTH ( string-expression )
```

Parameters

- **string-expression**  The string whose length is to be calculated.

Returns

`INT`

Remarks

Trailing white space characters are included in the length returned.

The return value of a NULL string is NULL.

If the string is in a multibyte character set, the value returned by the CHAR_LENGTH function may differ from the number of bytes returned by the BYTE_LENGTH function.

**Note**

You can use the CHAR_LENGTH function and the LENGTH function interchangeably for CHAR, VARCHAR, LONG VARCHAR, and NCHAR data types. However, you must use the LENGTH function for BINARY and bit array data types.

This function supports NCHAR inputs and/or outputs.

See also

- “BYTE_LENGTH function [String]” on page 171
- “String functions” on page 152

Standards and compatibility

- **SQL/2008**  CHAR_LENGTH is a core feature of the SQL/2008 standard. Using CHAR_LENGTH over an expression of type NCHAR comprises part of optional SQL/2008 language feature F421.

Example

The following statement returns the value 8:

```
SELECT CHAR_LENGTH( 'Chemical' );
```

CHARINDEX function [String]

Returns the position of one string in another.
Syntax

CHARINDEX( string-expression-1, string-expression-2 )

Parameters

- **string-expression-1**  The string for which you are searching.
- **string-expression-2**  The string to be searched.

Returns

INT

Remarks

The first character of *string-expression-1* is identified as 1. If the string being searched contains more than one instance of the other string, then the CHARINDEX function returns the position of the first instance.

If the string being searched does not contain the other string, then the CHARINDEX function returns 0.

If any of the arguments are NULL, the result is NULL.

This function supports NCHAR inputs and/or outputs.

See also

- “SUBSTRING function [String]” on page 379
- “REPLACE function [String]” on page 345
- “LOCATE function [String]” on page 287
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns last and first names from the Surname and GivenName columns of the Employees table, but only when the last name begins with the letter K:

```sql
SELECT Surname, GivenName
FROM GROUPO.Employees
WHERE CHARINDEX( 'K', Surname ) = 1;
```

Results returned:

<table>
<thead>
<tr>
<th>Surname</th>
<th>GivenName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klobucher</td>
<td>James</td>
</tr>
<tr>
<td>Kuo</td>
<td>Felicia</td>
</tr>
<tr>
<td>Kelly</td>
<td>Moira</td>
</tr>
</tbody>
</table>
COALESCE function [Miscellaneous]

Returns the first non-NULL expression from a list. This function is identical to the ISNULL function.

Syntax

COALESCE( expression, expression [ , ... ] )

Parameters

- expression  Any expression.

At least two expressions must be passed into the function, and all expressions must be comparable.

Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

Remarks

The result is NULL only if all the arguments are NULL.

The parameters can be of any scalar type, but not necessarily same type.

For a more detailed description of how the database server processes this function, see “ISNULL function [Miscellaneous]” on page 277.

See also

- “ISNULL function [Miscellaneous]” on page 277

Standards and compatibility

- SQL/2008  Core feature.

Example

The following statement returns the value 34:

    SELECT COALESCE( NULL, 34, 13, 0 );

COMPARE function [String]

Allows you to compare two character strings based on alternate collation rules.

Syntax

    COMPARE( string-expression-1, string-expression-2 [, { collation-id | collation-name[(collation-tailoring-string) ] } ] )


Parameters

- **string-expression-1**  The first string expression.

- **string-expression-2**  The second string expression.

  The string expression can only contain characters that are encoded in the database's character set.

- **collation-id**  A variable or integer constant that specifies the sort order to use. You can only use a `collation-id` for built-in collations. See “SORTKEY function [String]” on page 364.

  If you do not specify a collation name or ID, the default is Default Unicode multilingual.

- **collation-name**  A string or a character variable that specifies the name of the collation to use. You can also specify `char_collation` or `db_collation` (for example, `COMPARE ( 'abc', 'ABC', 'char_collation' );`) to use the database's CHAR collation. Similarly, you can specify `nchar_collation` to use the database's NCHAR collation. For a list of valid collation names, see “SORTKEY function [String]” on page 364.

- **collation-tailoring-string**  Optionally, you can specify collation tailoring options (`collation-tailoring-string`) for additional control over the character comparison. These options take the form of keyword=value pairs in parentheses, following the collation name. For example, `'UCA(locale=es;case=LowerFirst;accent=respect)'`. The syntax for specifying these options is identical to the syntax defined for the COLLATION clause of the CREATE DATABASE statement.

  **Note**
  All the collation tailoring options are supported when specifying the UCA collation. For all other collations, only case sensitivity tailoring option is supported.

See also

- “Collation tailoring options” [SQL Anywhere Server - Database Administration]

Returns

An INTEGER, based on the collation rules that you choose:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>string-expression-1</code> is greater than <code>string-expression-2</code></td>
</tr>
<tr>
<td>0</td>
<td><code>string-expression-1</code> is equal to <code>string-expression-2</code></td>
</tr>
<tr>
<td>-1</td>
<td><code>string-expression-1</code> is less than <code>string-expression-2</code></td>
</tr>
</tbody>
</table>

Remarks

The COMPARE function does not equate empty strings and strings containing only spaces, even if the database has blank-padding enabled. The COMPARE function uses the SORTKEY function to generate collation keys for comparison. Therefore, an empty string, a string with one space, and a string with two spaces do not compare equally.
If either string-expression-1 or string-expression-2 is NULL, the result is NULL.

See also
- “SORTKEY function [String]” on page 364
- “String functions” on page 152

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example performs three comparisons using the COMPARE function:

```
SELECT COMPARE( 'abc','ABC','UCA(case=LowerFirst)' ),
       COMPARE( 'abc','ABC','UCA(case=Ignore)' ),
       COMPARE( 'abc','ABC','UCA(case=UpperFirst)' );
```

The values returned are -1, 0, 1, indicating the result of each comparison. The first comparison results in -1, indicating that string-expression-2 ('ABC') is less than string-expression-1 ('abc'). This is because case sensitivity is set to LowerFirst in the first COMPARE statement.

### COMPRESS function [String]
Compresses the string and returns a value of type LONG BINARY.

**Syntax**
```
COMPRESS( string-expression [, 'compression-algorithm-alias' ])
```

**Parameters**
- **string-expression** The string to be compressed. Binary values can be passed to this function. This parameter is case sensitive, even in case-insensitive databases.
- **compression-algorithm-alias** Alias for the algorithm to use for compression. The supported values are 'zip' and 'gzip' (both are based on the same algorithm, but use different headers and trailers).

Zip is a widely supported compression algorithm. Gzip is compatible with the gzip utility on Unix, whereas the zip algorithm is not.

Decompression must be performed with the same algorithm.

**Returns**
LONG BINARY

**Remarks**
The value returned by the COMPRESS is not human-readable. If the value returned is longer than the original string, its maximum size will not be larger than a 0.1% increase over the original string + 12 bytes. You can decompress a compressed string-expression using the DECOMPRESS function.
If you are storing compressed values in a table, the column should be BINARY or LONG BINARY so that character set conversion is not performed on the data.

See also

- “DECOMPRESS function [String]” on page 219
- “String functions” on page 152

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example returns the length of the binary string created by compressing the string 'Hello World' using the gzip algorithm. This example can be useful when you want to determine whether a value has a shorter length when compressed.

```
SELECT LENGTH( COMPRESS( 'Hello world', 'gzip' ) );
```

**CONFLICT function [Miscellaneous]**

Indicates if a column is a source of conflict for an UPDATE being performed against a consolidated database in a SQL Remote environment.

Syntax

```
CONFLICT( column-name )
```

Parameters

- **column-name** The name of the column being tested for conflicts.

Returns

Returns TRUE if the column appears in the VERIFY list of an UPDATE statement executed by the SQL Remote Message Agent and if the value provided in the VALUES list of that statement does not match the original value of the column in the row being updated. Otherwise, returns FALSE.

See also

- “CREATE TRIGGER statement” on page 713
- “Default resolution for update conflicts” [SQL Remote]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The CONFLICT function is intended for use in SQL Remote RESOLVE UPDATE triggers to avoid error messages. To illustrate the use of the CONFLICT function, consider the following table:
CREATE TABLE Admin (
PKey bigint NOT NULL DEFAULT GLOBAL AUTOINCREMENT,
TextCol CHAR(20) NULL, PRIMARY KEY ( PKey ) );

Assume that consolidated and remote databases both have the following row in the Admin table:

1, 'Initial'

Now, at the consolidated database, update the row as follows:

UPDATE Admin SET TextCol = 'Consolidated Update' WHERE PKey = 1;

At the remote database, update the row to a different value as follows:

UPDATE Admin SET TextCol = 'Remote Update' WHERE PKey = 1;

Next, run dbremote on the remote database. It generates a message file with the following statements in it, to be executed at the consolidated database:

UPDATE Admin SET TextCol='Remote Update'
VERIFY ( TextCol )
VALUES ('Initial')
WHERE PKey=1;

When the SQL Remote Message Agent runs at the consolidated database and applies this UPDATE statement, SQL Anywhere uses the VERIFY and VALUES clause to determine whether a RESOLVE UPDATE trigger will fire. A RESOLVE UPDATE trigger fires only when the update is executed from the SQL Remote Message Agent against a consolidated database. Here is a RESOLVE UPDATE trigger:

CREATE TRIGGER ResolveUpdateAdmin
RESOLVE UPDATE ON Admin
REFERENCING OLD AS OldConsolidated
NEW AS NewRemote
REMOTE as OldRemote
FOR EACH ROW BEGIN
MESSAGE 'OLD';
MESSAGE OldConsolidated.PKey || ',' || OldConsolidated.TextCol;
MESSAGE 'NEW';
MESSAGE NewRemote.PKey || ',' || NewRemote.TextCol;
MESSAGE 'REMOTE';
MESSAGE OldRemote.PKey || ',' || OldRemote.TextCol;
END;

The RESOLVE UPDATE trigger fires because the current value of the TextCol column at the consolidated database ('Consolidated Update') does not match the value in the VALUES clause for the associated column ('Initial').

This trigger results in a failure because the PKey column was not modified in the UPDATE statement executed on the remote, so there is no OldRemote.PKey value accessible from this trigger.

The CONFLICT function helps to avoid this error by returning the following values:

- If there is no OldRemote.PKey value, return FALSE.
- If there is an OldRemote.PKey value, but it matches OldConsolidated.PKey, return FALSE.
If there is an OldRemote.PKey value, and it is different than OldConsolidated.PKey, return TRUE.

You can use the CONFLICT function to rewrite the trigger as follows and avoid the error:

```sql
CREATE TRIGGER ResolveUpdateAdmin
RESOLVE UPDATE ON Admin
REFERENCING OLD AS OldConsolidated
    NEW AS NewRemote
    REMOTE as OldRemote
FOR EACH ROW BEGIN
    message 'OLD';
    message OldConsolidated.PKey || ',' || OldConsolidated.TextCol;
    message 'NEW';
    message NewRemote.PKey || ',' || NewRemote.TextCol;
    message 'REMOTE';
    if CONFLICT( PKey ) then
        message OldRemote.PKey;
    end if;
    if CONFLICT( TextCol ) then
        message OldRemote.TextCol;
    end if;
END;
```

### CONNECTION_EXTENDED_PROPERTY function [String]

Returns the value of the given property. Allows an optional property-specific string parameter to be specified.

**Syntax**

```sql
CONNECTION_EXTENDED_PROPERTY(
    { property-id | property-name } |
    [ , property-specific-argument [, connection-id ] ] )
```

**Parameters**

- **property-id** The connection property ID.
- **property-name** The connection property name. The supported property names are:
  
  - **CharSet** Returns the CHAR character set label for the connection as it is known by the specified standard. The possible values include: ASE, IANA, MIME, JAVA, WINDOWS, UTR22, IBM, and ICU. The default is IANA unless the database connection was made through TDS in which case ASE is the default.
  
  - **NCharCharSet** Returns the NCHAR character set label for the connection as it is known by the specified standard. The possible values are the same as listed above for CharSet.
  
  - **Progress** Returns information about how long a statement has been executing. Specify a property-specific-argument, followed by connection-id, to return information specific to the statement's progress.
The following statements and procedures support the Progress property:

- BACKUP DATABASE (both image and archive)
- LOAD TABLE (USING FILE and USING CLIENT FILE only)
- MESSAGE
- REORGANIZE TABLE
- RESTORE DATABASE
- UNLOAD (all types)
- sa_table_page_usage system procedure

- **property-specific-argument** An optional property-specific string parameter associated with the following connection property:
  
  - **Progress**
    
    - **PercentComplete** Specify PercentComplete to obtain the percentage of the statement that has been processed.
    
    - **Completed** Specify Completed to obtain the completed number of units.
    
    - **Total** Specify Total to obtain the total number of units left to be processed.
    
    - **Units** Specify Units to obtain the type of units left to be processed (pages, rows, or bytes).
    
    - **Elapsed** Specify Elapsed to obtain the current elapsed time in milliseconds.
    
    - **Remaining** Specify Remaining to obtain the estimated remaining time in milliseconds.
    
    - **Raw** Specify Raw to obtain a string combining all of the above values in the order listed, separated by semicolons. For example, 43;9728;22230;pages;5025;6138.
    
    - **Formatted** Specify Formatted to obtain the human readable format. For example:

      ```
      43% ( 9728 of 22230 pages ) complete after 00:00:05; estimated 00:00:06 remaining
      ```

      The Remaining value may be empty if the remaining time has not yet been estimated, or if the number of units completed is greater than the original estimate.
      For all property-specific arguments except Formatted, large byte values are never converted to kilobytes or megabytes.

- **connection-id** The connection ID number of a database connection. The ID number for the current connection is used if a value is not specified.

**Returns**

Returns extended connection properties. The returned value is a VARCHAR.

**Remarks**

Either the property ID or the property name must be specified.
The CONNECTION_EXTENDED_PROPERTY function is similar to the CONNECTION_PROPERTY function except that it allows an optional property-specific string parameter to be specified. The interpretation of the property-specific argument depends on the property ID or name specified in the first argument.

You can use the CONNECTION_EXTENDED_PROPERTY function to return the value for any connection property. However, extended information is only available for the extended properties.

Cloud note: Because of tenant database isolation rules, when this function is called in the cloud it returns only information about the current tenant database.

Privileges
No privileges are required to execute this function for the current connection ID. To execute this function for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

NULL is returned if you specify an invalid parameter value or don't have one of the required system privileges.

See also
- “List of connection properties” [SQL Anywhere Server - Database Administration]
- “progress_messages option” [SQL Anywhere Server - Database Administration]
- “CONNECTION_PROPERTY function [System]” on page 186
- “DB_EXTENDED_PROPERTY function [System]” on page 212
- “DB_PROPERTY function [System]” on page 218

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example returns the CHAR character set of the current connection as it is known by the Java standard:

```sql
SELECT CONNECTION_EXTENDED_PROPERTY( 'charset', 'Java' );
```

CONNECTIONPROPERTY function [System]
Returns the value of a given connection property as a string.

Syntax
```
CONNECTION_PROPERTY(
    { property-id | property-name }
    [, connection-id ]
)
```

Parameters
- property-id The connection property ID.
● **property-name**  The connection property name.

● **connection-id**  The connection ID number of a database connection. The ID number for the current connection is used if a value is not specified.

Returns

VARCHAR, LONG VARCHAR

Remarks

Either the property ID or the property name must be specified.

⚠️ **Cloud note:** Because of tenant database isolation rules, when this function is called in the cloud it returns only information about the current tenant database.

Privileges

No privileges are required to execute this function for the current connection ID. To execute this function for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

NULL is returned if you specify an invalid parameter value or don't have one of the required system privileges.

See also

● “List of connection properties” [SQL Anywhere Server - Database Administration]

● “Accessing connection property values” [SQL Anywhere Server - Database Administration]

● “PROPERTY_NUMBER function [System]” on page 323

Standards and compatibility

● SQL/2008  Vendor extension.

Example

The following statement returns the number of prepared statements being maintained:

```sql
SELECT CONNECTION_PROPERTY( 'PrepStmt' );
```

**CONVERT function [Data type conversion]**

Returns an expression converted to a supplied data type.

**Syntax**

```
CONVERT( datatype, expression [, format-style ] )
```

**Parameters**

● **datatype**  The data type to which the expression is converted.

● **expression**  The expression to be converted.
- **format-style**  The style code to apply to the output value. Use this parameter when converting strings to date or time data types, and vice versa. The table below shows the supported style codes, followed by a representation of the output format produced by that style code. The style codes are separated into two columns, depending on whether the century is included in the output format (for example, 06 versus 2006).

Style code 0 is used if an argument is not provided.

<table>
<thead>
<tr>
<th>Without century (yy) style codes</th>
<th>With century (yyyy) style codes</th>
<th>Output format</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0 or 100</td>
<td>Mmm dd yyyy hh:nnAA</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>mm/dd/yy[yy]</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>[yy]yy.mm.dd</td>
</tr>
<tr>
<td>3</td>
<td>103</td>
<td>dd/mm/yy[yy]</td>
</tr>
<tr>
<td>4</td>
<td>104</td>
<td>dd.mm.yy[yy]</td>
</tr>
<tr>
<td>5</td>
<td>105</td>
<td>dd-mm-yy[yy]</td>
</tr>
<tr>
<td>6</td>
<td>106</td>
<td>dd Mmm yy[yy]</td>
</tr>
<tr>
<td>7</td>
<td>107</td>
<td>Mmm dd, yy[yy]</td>
</tr>
<tr>
<td>8</td>
<td>108</td>
<td>hh:nn:ss</td>
</tr>
<tr>
<td>-</td>
<td>9 or 109</td>
<td>Mmm dd yyyy hh:nn:ss:sssAA</td>
</tr>
<tr>
<td>10</td>
<td>110</td>
<td>mm-dd-yy[yy]</td>
</tr>
<tr>
<td>11</td>
<td>111</td>
<td>[yy]yy/mm/dd</td>
</tr>
<tr>
<td>12</td>
<td>112</td>
<td>[yy]yymmd</td>
</tr>
<tr>
<td>-</td>
<td>13 or 113</td>
<td>dd Mmm yyyy hh:nn:ss:sss (24 hour clock, Europe default + milliseconds, 4-digit year )</td>
</tr>
<tr>
<td>-</td>
<td>14 or 114</td>
<td>hh:nn:ss:sss (24 hour clock)</td>
</tr>
<tr>
<td>-</td>
<td>20 or 120</td>
<td>yyyy-mm-dd hh:nn:ss (24-hour clock, ODBC canonical, 4-digit year)</td>
</tr>
<tr>
<td>-</td>
<td>21 or 121</td>
<td>yyyy-mm-dd hh:nn:ss.sss (24 hour clock, ODBC canonical with milliseconds, 4-digit year)</td>
</tr>
</tbody>
</table>
Returns

Depends on the data type specified.

Remarks

The CONVERT function can be used to convert a string to a DATE, TIME, or TIMESTAMP data type, provided that there is no ambiguity when parsing the string. If format-style is specified, the database server may use it as a hint on how to parse the string. The database server returns an error if it cannot parse the string unambiguously.

For information about the styles produced by each output symbol (such as Mmm), see “date_format option” [SQL Anywhere Server - Database Administration].

See also

- “CAST function [Data type conversion]” on page 174
- “CSCONVERT function [String]” on page 198

Standards and compatibility

- SQL/2008  Vendor extension. The CONVERT function is defined in the SQL/2008 standard. However, in the SQL standard the purpose of CONVERT is to perform a transcoding of the input string expression to a different character set, which is implemented in SQL Anywhere as the CSCONVERT function.

Example

The following statements illustrate the use of format style:

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 104 ) FROM GROUPO.SalesOrders;
```

<table>
<thead>
<tr>
<th>OrderDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.03.2000</td>
</tr>
<tr>
<td>20.03.2000</td>
</tr>
<tr>
<td>23.03.2000</td>
</tr>
<tr>
<td>25.03.2000</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 7 ) FROM GROUPO.SalesOrders;
```

<table>
<thead>
<tr>
<th>OrderDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mar 16, 00</td>
</tr>
<tr>
<td>Mar 20, 00</td>
</tr>
</tbody>
</table>
The following statement illustrates conversion to an integer and returns the value 5:

```
SELECT CONVERT(integer, 5.2);
```

**CORR function [Aggregate]**

Returns the correlation coefficient of a set of number pairs.

**Syntax**

```
CORR(dependent-expression, independent-expression)
```

**Parameters**

- `dependent-expression` The variable that is affected by the independent variable.
- `independent-expression` The variable that influences the outcome.

**Returns**

DOUBLE

**Remarks**

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

Both `dependent-expression` and `independent-expression` are numeric. The function is applied to the set of `(dependent-expression, independent-expression)` after eliminating the pairs for which either `dependent-expression` or `independent-expression` is NULL. The following computation is made:

```
COVAR_POP(y, x) / STDDEV_POP(y) * STDDEV_POP(x)
```

where `y` represents the `dependent-expression` and `x` represents the `independent-expression`.

**See also**

- “Aggregate functions” on page 143
- “COVAR_POP function [Aggregate]” on page 195
- “STDDEV_POP function [Aggregate]” on page 372

**Standards and compatibility**

- **SQL/2008** The CORR function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".
Example
The following example performs a correlation to discover whether age is associated with income level and returns the value 0.44022675645996:

```
SELECT CORR( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) ) FROM GROUPO.Employees;
```

**COS function [Numeric]**

Returns the cosine of the angle in radians given by its argument.

**Syntax**

```
COS( numeric-expression )
```

**Parameters**

- `numeric-expression` The angle, in radians.

**Returns**

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

**See also**

- “ACOS function [Numeric]” on page 156
- “COT function [Numeric]” on page 191
- “SIN function [Numeric]” on page 362
- “TAN function [Numeric]” on page 385

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statement returns the value of the cosine of an angle 0.52 radians:

```
SELECT COS( 0.52 );
```

**COT function [Numeric]**

Returns the cotangent of the angle in radians given by its argument.

**Syntax**

```
COT( numeric-expression )
```

**Parameters**

- `numeric-expression` The angle, in radians.
Returns
This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

See also
- “COS function [Numeric]” on page 191
- “SIN function [Numeric]” on page 362
- “TAN function [Numeric]” on page 385

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement returns the cotangent value of 0.52:

```
SELECT COT( 0.52 );
```

COUNT function [Aggregate]
Counts the number of rows in a group depending on the specified parameters.

Syntax 1
```
COUNT( [ * | [ ALL | DISTINCT ] expression ] )
```

Syntax 2
```
COUNT( [ * | [ ALL ]expression ] ) OVER ( window-spec )
```

`window-spec`: see Syntax 2 instructions in the Remarks section below

Parameters
- `*` Return the number of rows in each group. COUNT(*) and COUNT() are semantically equivalent.
- `[ ALL ] expression` Return the number of rows in each group where the value of `expression` is not NULL.
- `DISTINCT expression` Return the number of distinct values of `expression` for all of the rows in each group where `expression` is not NULL.

Returns
The COUNT function returns a value of type INT.

COUNT never returns the value NULL. If a group contains no rows, or if there are no non-NULL values of `expression` in a group, then COUNT returns 0.

Remarks
The COUNT function returns a maximum value of 2147483647. Use the COUNT_BIG function when counting large result sets, the result might have more rows, or there is a possibility of overflow.
Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the `window-spec` definition in “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also
- “COUNT( * )” [SQL Anywhere Server - SQL Usage]
- “AVG function [Aggregate]” on page 164
- “SUM function [Aggregate]” on page 381
- “COUNT_BIG function [Aggregate]” on page 193
- “Troubleshooting: Aggregate functions and outer references” [SQL Anywhere 16 - Changes and Upgrading]

Standards and compatibility
- SQL/2008 Core feature. When used as a window function (Syntax 2), COUNT comprises part of optional SQL/2008 language feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional SQL language feature F561, "Full value expressions". SQL Anywhere also supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the COUNT function, combined with an outer reference.

For an example, see “AVG function [Aggregate]” on page 164.

Example
The following statement returns each unique city, and the number of employees working in that city:

```
SELECT City, COUNT( * ) FROM GROUPO.Employees GROUP BY City;
```

**COUNT_BIG function [Aggregate]**
Counts the number of rows in a group depending on the specified parameters.

**Syntax 1**
```
COUNT_BIG( [ * | [ ALL | DISTINCT ] expression ] )
```
Syntax 2

```
COUNT_BIG( [ * | [ ALL ] expression ] ) OVER ( window-spec )
```

*window-spec*: see Syntax 2 instructions in the Remarks section below

**Parameters**

- `*`  Return the number of rows in each group. COUNT_BIG(*) and COUNT_BIG() are semantically equivalent.

- `[ ALL ] expression`  Return the number of rows in each group where the value of `expression` is not NULL.

- `DISTINCT expression`  Return the number of distinct values of `expression` for all of the rows in each group where `expression` is not NULL.

**Returns**

COUNT_BIG returns a value of type BIGINT.

COUNT_BIG never returns the value NULL. If a group contains no rows, or if there are no non-NULL values of `expression` in a group, then COUNT_BIG returns 0.

**Remarks**

It is recommended that you use the COUNT_BIG function when counting large result sets, the result might have more rows, or there is a possibility of overflow. Otherwise, use the COUNT function, which has a maximum value of 2147483647.

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the `window-spec` definition in “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

**See also**

- “AVG function [Aggregate]” on page 164
- “SUM function [Aggregate]” on page 381
- “COUNT function [Aggregate]” on page 192

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the COUNT_BIG function, combined with an outer reference.
For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

See also
- “Troubleshooting: Aggregate functions and outer references” [SQL Anywhere 16 - Changes and Upgrading]

Example
The following statement returns each unique city, and the number of employees working in that city.

```sql
SELECT City, COUNT_BIG( * ) FROM GROUPO.Employees GROUP BY City;
```

### COUNT_SET_BITS function [Bit array]
Returns a count of the number of bits set to 1 (TRUE) in the array.

**Syntax**

```
COUNT_SET_BITS( bit-expression )
```

**Parameters**

- `bit-expression` The bit array for which to determine the set bits.

**Returns**

UNSIGNED INT

**Remarks**

Returns NULL if `bit-expression` is NULL.

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statement returns the value 4:

```sql
SELECT COUNT_SET_BITS( '00110011' );
```

The following statement returns the value 12:

```sql
SELECT COUNT_SET_BITS( '0011001111111111' );
```

### COVAR_POP function [Aggregate]
Returns the population covariance of a set of number pairs.

**Syntax 1**

```
COVAR_POP( dependent-expression, independent-expression )
```
Syntax 2

COVAR_POP( dependent-expression, independent-expression )
OVER ( window-spec )

window-spec : see Syntax 2 instructions in the Remarks section below

Parameters

- **dependent-expression** The variable that is affected by the independent variable.
- **independent-expression** The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

Both dependent-expression and independent-expression are numeric. The function is applied to the set of (dependent-expression, independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The following computation is then made:

\[
\frac{\left( \text{SUM}(y \times x) - \text{SUM}(x) \times \text{SUM}(y) / n \right)}{n}
\]

where \(y\) represents the dependent-expression and \(x\) represents the independent-expression.

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition provided in “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also

- “COVAR_SAMP function [Aggregate]” on page 197
- “SUM function [Aggregate]” on page 381

Standards and compatibility

- **SQL/2008** The COVAR_POP function comprises part of optional SQL/2008 language feature T621, “Enhanced numeric functions”.


Example
The following example measures the strength of association between employees' age and salary. This function returns the value 73785.84005866687.

```
SELECT COVAR_POP(Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

**COVAR_SAMP function [Aggregate]**

Returns the sample covariance of a set of number pairs.

**Syntax 1**

```
COVAR_SAMP( dependent-expression, independent-expression )
```

**Syntax 2**

```
COVAR_SAMP( dependent-expression, independent-expression )
OVER ( window-spec )
```

`window-spec`: see Syntax 2 instructions in the Remarks section below

**Parameters**

- `dependent-expression` The variable that is affected by the independent variable.
- `independent-expression` The variable that influences the outcome.

**Returns**

DOUBLE

**Remarks**

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

Both `dependent-expression` and `independent-expression` are numeric. The function is applied to the set of `(dependent-expression, independent-expression)` pairs after eliminating all pairs for which either `dependent-expression` or `independent-expression` is NULL.

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the `window-spec` definition provided in “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].
See also

- “COVAR_POP function [Aggregate]” on page 195
- “SUM function [Aggregate]” on page 381

Standards and compatibility

- **SQL/2008** The COVAR_SAMP function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions”.

Example

The following example returns the value 74782.9460054052.

```sql
SELECT COVAR_SAMP( Salary, ( 2008 - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

**CSCONVERT function [String]**

Converts strings between character sets.

Syntax

```sql
CSCONVERT(
    string-expression,
    target-charset-string [, source-charset-string [, options ] ])
```

Parameters

- **string-expression** The string to be converted.

- **target-charset-string** The destination character set. `target-charset-string` can be any of the SQL Anywhere supported character set labels. It can also be:
  - `os_charset` Specify this to use the character set used by the operating system that is hosting the database server.
  - `char_charset` Specify this to use the CHAR character set used by the database.
  - `nchar_charset` Specify this to use the NCHAR character set used by the database.
  - `options` You can specify one of the following options:
    - **Read or write a byte order mark (BOM)** Specify `read_bom=on` or `read_bom=off` to turn on or off reading byte order marks. Specify `write_bom=on` or `write_bom=off` to turn on or off writing byte order marks. By default, the behavior is `read_bom=on` and `write_bom=off`.

- **source-charset-string** The character set used for `string-expression`. The default is `db_charset` (the database character set). `source-charset-string` can be any of the SQL Anywhere supported character set labels. It can also be:
  - `os_charset` Specify this to use the character set used by the operating system that is hosting the database server.
- **char_charset** Specify this to use the CHAR character set used by the database.

- **nchar_charset** Specify this to use the NCHAR character set used by the database.

### Returns

**LONG BINARY**

### Remarks

You can view the list of character sets supported by SQL Anywhere by running the following command:

```
dbinit -le
```

For more information about the character set labels you can use with this function, see “Supported character sets” [*SQL Anywhere Server - Database Administration*].

### See also

- “String functions” on page 152

### Standards and compatibility

- **SQL/2008** Vendor extension. In the SQL/2008 standard, conversion of string data from one charset to another is accomplished with the CONVERT function (not to be confused with the SQL Anywhere CONVERT function) which has different arguments than CSCONVERT.

### Examples

This fragment converts the mytext column from the Traditional Chinese character set to the Simplified Chinese character set:

```sql
SELECT CSCONVERT( mytext, 'cp936', 'cp950' )
FROM mytable;
```

This fragment converts the mytext column from the database character set to the Simplified Chinese character set:

```sql
SELECT CSCONVERT( mytext, 'cp936' )
FROM mytable;
```

If a file name is stored in the database, it is stored in the database character set. If the server will read from or write to a file whose name is stored in a database (for example, in an external stored procedure), the file name must be explicitly converted to the operating system character set before the file can be accessed. File names stored in the database and retrieved by the client are converted automatically to the client character set, so explicit conversion is not necessary.

This fragment converts the value in the filename column from the database character set to the operating system character set:

```sql
SELECT CSCONVERT( filename, 'os_charset' )
FROM mytable;
```

A table contains a list of file names. An external stored procedure takes a file name from this table as a parameter and reads information directly out of that file. The following statement works when character set conversion is not required:
SELECT MYFUNC( filename )
FROM mytable;

The mytable clause indicates a table with a filename column. However, if you need to convert the file
name to the character set of the operating system, you would use the following statement.

SELECT MYFUNC( csconvert( filename, 'os_charset' ) )
FROM mytable;

**CUME_DIST function [Ranking]**

Computes the relative position of one value among a group of rows.

**Syntax**

```
CUME_DIST( ) OVER ( window-spec )
```

`window-spec`: see the Remarks section below

**Returns**

A DOUBLE value between 0 and 1

**Remarks**

Composite sort keys are not currently allowed in the CUME_DIST function. You can use composite sort
keys with any of the other rank functions.

Elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW
clause in the SELECT statement. When used as a window function, you must specify an ORDER BY
clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE
clause. See the `window-spec` definition provided in “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples,
see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window
definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

**See also**

- “DENSE_RANK function [Ranking]” on page 222
- “PERCENT_RANK function [Ranking]” on page 316
- “RANK function [Ranking]” on page 326

**Standards and compatibility**

- **SQL/2008** The CUME_DIST function comprises part of optional SQL/2008 language feature
  T612, "Advanced OLAP operations".

**Example**

The following example returns a result set that provides a cumulative distribution of the salaries of
employees who live in California:
SELECT DepartmentID, Surname, Salary, 
CUME_DIST() OVER (PARTITION BY DepartmentID 
ORDER BY Salary DESC) "Rank"
FROM GROUPO.Employees
WHERE State IN ('CA');

Here is the result set:

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>Surname</th>
<th>Salary</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Savarino</td>
<td>72300.000</td>
<td>0.333333333333333</td>
</tr>
<tr>
<td>200</td>
<td>Clark</td>
<td>45000.000</td>
<td>0.666666666666667</td>
</tr>
<tr>
<td>200</td>
<td>Overbey</td>
<td>39300.000</td>
<td>1</td>
</tr>
</tbody>
</table>

**DATALENGTH function [System]**

Returns the length, in bytes, of the underlying storage for the result of an expression.

**Syntax**

`DATALENGTH( expression )`

**Parameters**

- `expression` Usually a column name. If `expression` is a string constant, you must enclose it in quotes.

**Returns**

UNSIGNED INT

**Remarks**

The return values of the DATALENGTH function are as follows:

<table>
<thead>
<tr>
<th>Data type</th>
<th>DATALENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>1</td>
</tr>
<tr>
<td>TINYINT</td>
<td>1</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>BIGINT</td>
<td>8</td>
</tr>
<tr>
<td>REAL</td>
<td>4</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>8</td>
</tr>
</tbody>
</table>
### Data type | DATALENGTH
---|---
TIME | 8
DATE | 4
TIMESTAMP | 8
DATETIME | 8
TIMESTAMP WITH TIME ZONE | 29
UNIQUEIDENTIFIER | 16
CHAR | Length of the data
VARCHAR | Length of the data
BINARY | Length of the data
VARBINARY | Length of the data
NCHAR | Length of the data
NVARCHAR | Length of the data
TEXT | Length of the data
NTEXT | Length of the data
IMAGE | Length of the data
XML | Length of the data

This function supports NCHAR inputs and outputs.

**Standards and compatibility**
- SQL/2008 Vendor extension.

**See also**
- “SQL data types” on page 89

**Example**
The following statement returns the length of the longest string in the CompanyName column:

```sql
SELECT MAX( DATALENGTH( CompanyName ) )
FROM GROUPO.Customers;
```

The following statement returns the length of the string '8sdoinsv8s7a7s7gehe4h':
DATE function [Date and time]

Converts the expression into a date, and removes any hours, minutes, or seconds.

For information about controlling the interpretation of date formats, see “date_order option” [SQL Anywhere Server - Database Administration] and “nearest_century option” [SQL Anywhere Server - Database Administration].

Syntax

```
DATE( expression )
```

Returns

```
DATE
```

Parameters

- **expression** The value to be converted to date format, typically a string.

See also

- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMMP special value” on page 70
- “DATE data type” on page 115
- “DATETIME data type” on page 116
- “DATETIME function [Date and time]” on page 209
- “DATETIMEOFFSET data type” on page 117
- “Expressions” on page 21
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP special value” on page 76
- “TIMESTAMP data type” on page 121
- “UTC TIMESTAMMP special value” on page 78

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following statement returns the value 1999-01-02 as a date:

```
SELECT DATE( '1999-01-02 21:20:53' );
```

The following statement returns the create dates of all the objects listed in the SYSOBJECT system view:

```
SELECT DATE( creation_time ) FROM SYSOBJECT;
```
DATEADD function [Date and time]

Returns a TIMESTAMP or TIMESTAMP WITH TIME ZONE value produced by adding a date part to its argument.

Syntax

```
DATEADD( date-part, integer-expression, timestamp-expression )
```

date-part:

- year
- quarter
- month
- week
- day
- dayofyear
- hour
- minute
- second
- millisecond
- microsecond

Parameters

- **date-part**  The date part that `integer-expression` represents.
  
  For a complete listing of allowed date parts, see “Specifying date parts” on page 146.

- **integer-expression**  The number of date-part values to be added to timestamp-expression. `integer-expression` can be any numeric type, but its value is truncated to an INTEGER.

- **timestamp-expression**  The TIMESTAMP or TIMESTAMP WITH TIME ZONE value to be modified.

Returns

TIMESTAMP WITH TIME ZONE if `timestamp-expression` is a TIMESTAMP WITH TIME ZONE; otherwise TIMESTAMP.

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statement returns the TIMESTAMP value 1995-11-02 00:00:00.000:

```
SELECT DATEADD( month, 102, '1987/05/02' );
```

The following statement returns the TIMESTAMP value 1987-05-02 04:00:00.000:

```
SELECT DATEADD( hour, 4, '1987/05/02' );
```

The following statement returns the TIMESTAMP WITH TIME ZONE value 1987-05-06 11:33:00.000+04:00:
DATEDIFF function [Date and time]

Returns the interval between two dates.

Syntax

DATEDIFF( date-part, date-expression-1, date-expression-2 )

date-part:

year
quarter
month
week
day
dayofyear
hour
minute
second
millisecond
microsecond

Parameters

- **date-part** Specifies the date part in which the interval is to be measured.

  Choose one of the date objects listed above. For a complete list of date parts, see “Specifying date parts” on page 146.

- **date-expression-1** The starting date for the interval. This value is subtracted from date-expression-2 to return the number of date-parts between the two arguments.

- **date-expression-2** The ending date for the interval. Date-expression-1 is subtracted from this value to return the number of date-parts between the two arguments.

Returns

INT with year, quarter, month, week, day, and dayofyear. BIGINT with hour, minute, second, millisecond, and microsecond.

Remarks

This function calculates the number of date parts between two specified dates. The result is a signed integer value equal to (date-expression-2 - date-expression-1), in date parts.

The DATEDIFF function results are truncated, not rounded, when the result is not an even multiple of the date part.

When you use day as the date part, the DATEDIFF function returns the number of midnights between the two times specified, including the second date but not the first.
When you use **month** as the date part, the DATEDIFF function returns the number of first-of-the-months between two dates, including the second date but not the first.

When you use **week** as the date part, the DATEDIFF function returns the number of Sundays between the two dates, including the second date but not the first.

**Standards and compatibility**
- **SQL/2008** Vendor extension.

**Example**

The following statement returns 1:

```
SELECT DATEDIFF( hour, '4:00AM', '5:50AM' );
```

The following statement returns 102:

```
SELECT DATEDIFF( month, '1987/05/02', '1995/11/15' );
```

The following statement returns 0:

```
SELECT DATEDIFF( day, '00:00', '23:59' );
```

The following statement returns 4:

```
SELECT DATEDIFF( day, '1999/07/19 00:00', '1999/07/23 23:59' );
```

The following statement returns 0:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/07/23' );
```

The following statement returns 1:

```
SELECT DATEDIFF( month, '1999/07/19', '1999/08/23' );
```

---

**DATEFORMAT function [Date and time]**

Returns a string representing a date expression in the specified format.

**Syntax**

```
DATEFORMAT( datetime-expression, string-expression )
```

**Parameters**

- **datetime-expression** The datetime to be converted.
- **string-expression** The format of the converted date.

For information about date format descriptions, see “timestamp_format option” *SQL Anywhere Server - Database Administration*.

This function supports NCHAR inputs and/or outputs.
Returns
    VARCHAR

Remarks
    Any allowable date format can be used for the string-expression.

Standards and compatibility
    ● SQL/2008    Vendor extension.

Example
    The following statement returns the value Jan 01, 1989:

        SELECT DATEFORMAT( '1989-01-01', 'Mmm dd, yyyy' );

DATENAME function [Date and time]
    Returns the name of the specified part (such as the month June) of a TIMESTAMP or TIMESTAM
    P WITH TIME ZONE value, as a character string.

Syntax
    DATENAME( date-part, timestamp-expression )

Parameters
    ● date-part    The date part to be named.

        For a complete listing of allowed date parts, see “Specifying date parts” on page 146.
        
    ● timestamp-expression    The TIMESTAMP or TIMESTAMP WITH TIME ZONE value for which
                               the date part name is to be returned. For meaningful results, timestamp-expressi
                               on should contain the requested date-part.

Returns
    VARCHAR

Remarks
    The DATENAME function returns a string, even if the result is numeric, such as 23 for the day. W
    hen the date part TZOffset is specified, DATENAME returns the offset as a string of the form: { + | - }hh:nn.

See also
    ● “DATEPART function [Date and time]” on page 208

Standards and compatibility
    ● SQL/2008    Vendor extension.
Example

The following statement returns the value May:

```sql
SELECT DATENAME(month, '1987/05/02');
```

**DATEPART function [Date and time]**

Returns a portion of a TIMESTAMP or TIMESTAMP WITH TIME ZONE value.

**Syntax**

```
DATEPART( date-part, timestamp-expression )
```

**Parameters**

- `date-part`  The date part to be returned.
  
  For a complete listing of allowed date parts, see “Specifying date parts” on page 146.

- `timestamp-expression`  The TIMESTAMP or TIMESTAMP WITH TIME ZONE value for which the part is to be returned.

**Returns**

**INT**

**Remarks**

For meaningful results `timestamp-expression` should contain the required `date-part` portion.

The numbers that correspond to week days depend on the setting of the first_day_of_week option. By default Sunday=7.

**See also**

- “first_day_of_week option” [SQL Anywhere Server - Database Administration]
- “SET statement [T-SQL]” on page 987

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

**Example**

The following statement returns the value 5:

```sql
SELECT DATEPART(month, '1987/05/02');
```

The following example creates a table, TableStatistics, and inserts into it the total number of sales orders per year as stored in the SalesOrders table:

```sql
CREATE TABLE TableStatistics (  
    ID INTEGER NOT NULL DEFAULT AUTOINCREMENT,  
    Year INT,  
    NumberOrders INT );
```
INSERT INTO TableStatistics ( Year, NumberOrders )
    SELECT DATEPART( Year, OrderDate ), COUNT(*)
    FROM GROUP0.SalesOrders
    GROUP BY DATEPART( Year, OrderDate );

DATETIME function [Date and time]

Converts an expression into a TIMESTAMP value.

Syntax

    DATETIME( expression )

Parameters

   • expression     The expression to be converted. It is generally a string.

Returns

    TIMESTAMP

Remarks

    Attempts to convert numerical values return an error.

See also

   • “Expressions” on page 21
   • “CAST function [Data type conversion]” on page 174
   • “CURRENT TIME special value” on page 67
   • “CURRENT TIMESTAMP special value” on page 68
   • “CURRENT UTC TIMESTAMP special value” on page 70
   • “DATE data type” on page 115
   • “DATE function [Date and time]” on page 203
   • “DATETIME data type” on page 116
   • “DATETIMEOFFSET data type” on page 117
   • “GETDATE function [Date and time]” on page 251
   • “ISDATE function [Data type conversion]” on page 275
   • “NOW function [Date and time]” on page 311
   • “SMALLDATETIME data type” on page 119
   • “TIME data type” on page 120
   • “TIMESTAMP special value” on page 76
   • “TIMESTAMP data type” on page 121
   • “UTC TIMESTAMP special value” on page 78

Standards and compatibility

   • SQL/2008      Vendor extension.

Example

    The following statement returns a timestamp with value 1998-09-09 12:12:12.000:

    SELECT DATETIME( '1998-09-09 12:12:12.000' );
DAY function [Date and time]
Returns the day of the month of its argument as an integer between 1 and 31.

Syntax
```sql
DAY( date-expression )
```

Parameters
- `date-expression` The date as a DATE data type.

Returns
`SMALLINT`

Remarks
The DAY function returns an integer between 1 and 31, corresponding to the day of the month in the argument.

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement returns the value 12:
```sql
SELECT DAY( '2001-09-12' );
```

DAYNAME function [Date and time]
Returns the name of the day of the week from a date.

Syntax
```sql
DAYNAME( date-expression )
```

Parameters
- `date-expression` The date.

Returns
`VARCHAR`

Remarks
The English names are returned as: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday.

Standards and compatibility
- SQL/2008 Vendor extension.
Example

The following statement returns the value Saturday:

```
SELECT DAYNAME ( '1987/05/02' );
```

### DAYS function [Date and time]
Manipulates a TIMESTAMP or returns the number of days between two TIMESTAMP values. For specific details, see the Remarks section below.

#### Syntax 1

```
DAYS( timestamp-expression )
```

#### Syntax 2

```
DAYS( timestamp-expression, timestamp-expression )
```

#### Syntax 3

```
DAYS( timestamp-expression, integer-expression )
```

#### Parameters
- **timestamp-expression** A TIMESTAMP value.
- **integer-expression** The number of days to be added to the timestamp-expression. If the integerexpression is negative, the appropriate number of days is subtracted from timestamp-expression. If you supply an integer expression, the timestamp-expression must be explicitly cast as a TIME, DATE or TIMESTAMP. If timestamp-expression is a TIME value, the current date is assumed.

For information about casting data types, see “CAST function [Data type conversion]” on page 174.

#### Returns
- INTEGER with Syntax 1 or Syntax 2.
- TIMESTAMP with Syntax 3.

#### Remarks
The result of the DAYS function depends on its arguments. The DAYS function ignores hours, minutes, and seconds in its arguments.

- **Syntax 1** If you pass a single timestamp-expression to the DAYS function, it will return the number of days between 0000-02-29 and timestamp-expression as an INTEGER.

```
Note
0000-02-29 is not meant to imply an actual date; it is the default date used by the DAYS function.
```

- **Syntax 2** If you pass two TIMESTAMP values to the DAYS function, the function returns the integer number of days between them.
• **Syntax 3**  If you pass a TIMESTAMP value and an integer to the DAYS function, the function returns the TIMESTAMP result of adding the integer number of days to the *timestamp-expression* argument.

Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.

**See also**
- “DATEDIFF function [Date and time]” on page 205
- “DATEADD function [Date and time]” on page 204

**Standards and compatibility**
- **SQL/2008**  Vendor extension.

**Example**
The following statement returns the integer 729889:

```sql
SELECT DAYS( '1998-07-13 06:07:12' );
```

The following statements return the integer value -366, indicating that the second DATE value is 366 days before the first. It is recommended that you use the second example (DATEDIFF):

```sql
SELECT DAYS( '1998-07-13 06:07:12',
             '1997-07-12 10:07:12' );

SELECT DATEDIFF( day,
                 '1998-07-13 06:07:12',
                 '1997-07-12 10:07:12' );
```

The following statements return the TIMESTAMP value 1999-07-14 00:00:00.000. It is recommended that you use the second example (DATEADD):

```sql
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 );
SELECT DATEADD( day, 366, '1998-07-13' );
```

**DB_EXTENDED_PROPERTY function [System]**
Returns the value of the given property. Allows an optional property-specific string parameter to be specified.

**Syntax**

```sql
DB_EXTENDED_PROPERTY(
    { property-id | property-name }
    [, property-specific-argument
      [, database-id | database-name ] ]
)
```

**Parameters**
- **property-id**  The database property ID to query.
• **property-name**  The database property name to query.

For a complete list of database properties, see “List of database properties” [SQL Anywhere Server - Database Administration].

• **property-specific-argument**  The following database properties allow you to specify additional arguments, as noted below, to return specific information about the property.

  ○ **CharSet property**  Specify the name of a standard to obtain the default CHAR character set label for the standard. Possible values you can specify are: ASE, IANA, MIME, JAVA, WINDOWS, UTR22, IBM, and ICU. If no standard is specified, IANA is used as the default, unless the database connection was made through TDS, in which case ASE is the default.

  ○ **CatalogCollation, Collation, and NcharCollation properties**  When querying these properties, the following values can be specified as a `property-specific-argument` to return information specific to the collation:

    • **AccentSensitive**  Specify AccentSensitive to obtain the accent sensitivity setting for the collation. For example, the following statement returns the accent sensitivity setting for the NCHAR collation:

      ```
      SELECT DB_EXTENDED_PROPERTY( 'NcharCollation', 'AccentSensitive');
      ```

      Possible return values are: Ignore, Respect, and French. For more information, see “Collation tailoring options” [SQL Anywhere Server - Database Administration].

    • **CaseSensitivity**  Specify CaseSensitivity to obtain the case sensitivity setting for the collation. Possible return values are: Ignore, Respect, UpperFirst, and LowerFirst. For more information, see “Collation tailoring options” [SQL Anywhere Server - Database Administration].

    • **PunctuationSensitivity**  Specify PunctuationSensitivity to obtain the punctuation sensitivity setting for the collation. Possible return values are: Ignore, Primary, and Quaternary. For more information, see “Collation tailoring options” [SQL Anywhere Server - Database Administration].

    • **Properties**  Specify Properties to obtain a string containing all the tailoring options specified for the collation. For information about the keywords and values in the returned string, see “Collation tailoring options” [SQL Anywhere Server - Database Administration].

    • **Specification**  Specify Specification to obtain a string containing the full collation specification used for the collation. For information about the keywords and values in the returned string, see “Collation tailoring options” [SQL Anywhere Server - Database Administration].

  ○ **DriveType property**  Specify the name of a dbspace, or the file ID for the dbspace, to obtain its drive type. The value returned is one of the following: CD, FIXED, RAMDISK, REMOTE, REMOVABLE, or UNKNOWN. If nothing is specified, the drive type of the system dbspace is returned. If the specified dbspace doesn't exist, the property function returns NULL. If the name of a dbspace is specified and the ID of a database that isn't the database of the current connection is also specified, the function also returns NULL.
○ **File property**  Specify a dbspace name to obtain the file name of the database root file, including the path. If nothing is specified, information for the system dbspace is returned. If the specified file doesn't exist, the function returns NULL.

○ **FileSize property**  Specify the name of a dbspace, or the file ID for the dbspace, to obtain the size of the specified file in pages. You can also specify temporary to return the size of the temporary dbspace, or translog to return the size of the log file. If nothing is specified, the size of the system dbspace is returned. If the specified file doesn't exist, the function returns NULL.

○ **FreePages property**  Specify the name of a dbspace, or the file ID for the dbspace, to obtain the number of free pages. You can also specify temporary to return the number of free pages in the temporary dbspace, or translog to return the number of free pages in the log file. If nothing is specified, the number of free pages in the system dbspace is returned. If the specified file doesn't exist, the function returns NULL.

○ **IOParallelism property**  Specify a dbspace name to obtain the estimated number of simultaneous I/O operations supported by the dbspace. If a dbspace is not specified, the current system dbspace is used.

○ **MirrorServerState property**  Specify a server name to determine the connection status of the mirror server. Returns connected, disconnected, incoming only, outgoing only, or NULL.

○ **MirrorState property**  Specify a server name to determine the synchronization status of the mirror server. Returns synchronizing, synchronized, or NULL.

○ **NextScheduleTime property**  Specify an event name to obtain its next scheduled execution time.

- **database-id**  The database ID number, as returned by the DB_ID function. Typically, the database name is used.

- **database-name**  The name of the database, as returned by the DB_NAME function.

**Returns**

VARCHAR

**Remarks**

The DB_EXTENDED_PROPERTY function is similar to the DB_PROPERTY function except that it allows an optional *property-specific-argument* string parameter to be specified. The interpretation of *property-specific-argument* depends on the property ID or name specified in the first argument.

The current database is used if the third argument is omitted.

When comparing catalog strings such as table names and procedure names, the database server uses the CHAR collation. For the UCA collation, the catalog collation is the same as the CHAR collation but with the tailoring changed to be case-insensitive, accent-insensitive and with punctuation sorted in the primary level. For legacy collations, the catalog collation is the same as the CHAR collation but with the tailoring changed to be case-insensitive. While you cannot explicitly specify the tailoring used for the catalog collation, you can query the Specification property to obtain the full collation specification used by the
database server for comparing catalog strings. Querying the Specification property can be useful if you need to exploit the difference between the CHAR and catalog collations. For example, suppose you have a punctuation-insensitive CHAR collation and you want to execute an upgrade script that defines a procedure called my_procedure, and that also attempts to delete an old version named myprocedure. The following statements cannot achieve the desired results because my_procedure is equivalent to myprocedure, using the CHAR collation:

```sql
CREATE PROCEDURE my_procedure( ) ...;
IF EXISTS ( SELECT * FROM SYS.SYSPROCEDURE WHERE proc_name = 'myprocedure' )
THEN DROP PROCEDURE myprocedure
END IF;
```

Instead, you could execute the following statements to achieve the desired results:

```sql
CREATE PROCEDURE my_procedure( ) ...;
IF EXISTS ( SELECT * FROM SYS.SYSPROCEDURE
WHERE COMPARE( proc_name, 'myprocedure',
DB_EXTENDED_PROPERTY( 'CatalogCollation', 'Specification' ) ) = 0 )
THEN DROP PROCEDURE myprocedure
END IF;
```

**Cloud note:** Because of tenant database isolation rules, when this function is called in the cloud it returns only information about the current tenant database.

**Privileges**

No privileges are required to execute this function for the current database. To execute this function for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

NULL is returned if you specify an invalid parameter value or don't have one of the required system privileges.

**See also**

- “DB_ID function [System]” on page 216
- “DB_NAME function [System]” on page 217
- “List of database properties” [SQL Anywhere Server - Database Administration]
- “CONNECTIONPROPERTY function [System]” on page 186
- “CONNECTION_EXTENDEDPROPERTY function [String]” on page 184

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statement returns the location of the current database:

```sql
SELECT DB_EXTENDED_PROPERTY( 'File' );
```

The following statement returns the file size of the system dbspace, in pages:

```sql
SELECT DB_EXTENDED_PROPERTY( 'FileSize' );
```

The following statement returns the file size of the transaction log, in pages:
The following statement returns the case sensitivity setting for the NCHAR collation:

```sql
SELECT DB_EXTENDED_PROPERTY( 'NcharCollation', 'CaseSensitivity' );
```

The following statement returns the tailoring options specified for the database CHAR collation:

```sql
SELECT DB_EXTENDED_PROPERTY ( 'Collation', 'Properties' );
```

The following statement returns the full collation specification for the database NCHAR collation:

```sql
SELECT DB_EXTENDED_PROPERTY( 'NcharCollation', 'Specification' );
```

The following statement returns the connection status of the mirror server Test:

```sql
SELECT DB_EXTENDED_PROPERTY( 'MirrorServerState', 'Test' );
```

The following statement returns the synchronization status of the mirror server Test:

```sql
SELECT DB_EXTENDED_PROPERTY( 'MirrorState', 'Test' );
```

## DB_ID function [System]

Returns the database ID number.

### Syntax

```sql
DB_ID([ database-name ])
```

### Parameters

- **database-name**
  
  A string containing the database name. If no `database-name` is supplied, the ID number of the current database is returned.

### Returns

INT

### Remarks

- **Cloud note:** Because of tenant database isolation rules, when this function is called in the cloud it returns only information about the current tenant database.

### Privileges

None

### See also

- “global_database_id option” [SQL Anywhere Server - Database Administration]

### Standards and compatibility

- **SQL/2008**  Vendor extension.
Example
The following statement returns the value 0, when executed against the SQL Anywhere sample database as the sole database on the server:

```
SELECT DB_ID( 'demo' );
```

The following statement returns the value 0 if executed against the only running database:

```
SELECT DB_ID( );
```

**DB_NAME function [System]**
Returns the name of a database with a given ID number.

**Syntax**
```
DB_NAME( [ database-id ] )
```

**Parameters**
- `database-id` The ID of the database. The `database-id` must be a numeric expression.

**Returns**
VARCHAR

**Remarks**
If no database ID is supplied, the name of the current database is returned.

*Cloud note:* Because of tenant database isolation rules, when this function is called in the cloud it returns only information about the current tenant database.

**Privileges**
No privileges are required to execute this function for the current database. To execute this function for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

**See also**
- [“sa_db_list system procedure” on page 1119](#)
- [“NEXT_DATABASE function [System]” on page 306](#)

**Standards and compatibility**
- SQL/2008 Vendor extension.

**Example**
The following statement returns the database name demo when executed against the SQL Anywhere sample database as the sole database on the server:

```
SELECT DB_NAME( 0 );
```
**DB_PROPERTY function [System]**

Returns the value of the given property.

**Syntax**

```
DB_PROPERTY(
    { property-id | property-name }
    [, database-id | database-name ]
)
```

**Parameters**

- **property-id**  The database property ID.
- **property-name**  The database property name.
- **database-id**  The database ID number, as returned by the DB_ID function. Typically, the database name is used.
- **database-name**  The name of the database, as returned by the DB_NAME function.

**Returns**

VARCHAR, LONG VARCHAR

**Remarks**

Returns a string. The current database is used if the second argument is omitted.

* Cloud note: Because of tenant database isolation rules, when this function is called in the cloud it returns only information about the current tenant database.

**Privileges**

No privileges are required to execute this function for the current database. To execute this function for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

NULL is returned if you specify an invalid parameter value or don't have one of the required system privileges.

**See also**

- “DB_ID function [System]” on page 216
- “DB_NAME function [System]” on page 217
- “List of database properties” [SQL Anywhere Server - Database Administration]
- “PROPERTY function [System]” on page 320

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

The following statement returns the page size of the current database, in bytes:
DECOMPRESS function [String]

Decompresses the string and returns a LONG BINARY value.

Syntax

```
DECOMPRESS( string-expression [, compression-algorithm-alias ] )
```

Parameters

- **string-expression**  The string to decompress. Binary values can also be passed to this function. This parameter is case sensitive, even in case-insensitive databases.
- **compression-algorithm-alias**  Alias (string) for the algorithm to use for decompression. The supported values are 'zip' and 'gzip' (both are based on the same algorithm, but use different headers and trailers).

Zip is a widely supported compression algorithm. Gzip is compatible with the gzip utility on Unix, whereas the zip algorithm is not.

If no algorithm is specified, the function attempts to detect which algorithm was used to compress the string. If the incorrect algorithm is specified, or the correct algorithm cannot be detected, the string is not decompressed.

For more information about compression, see “COMPRESS function [String]” on page 181.

Returns

LONG BINARY

Remarks

This function can be used to decompress a value that was compressed using the COMPRESS function.

You do not need to use the DECOMPRESS function on values that are stored in a compressed column. Compression and decompression of values in a compressed column are handled automatically by the database server.

See also

- “Column compression considerations” [SQL Anywhere Server - Database Administration]
- “COMPRESS function [String]” on page 181
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example uses the DECOMPRESS function to decompress values from the Attachment column of a fictitious table, TableA:
SELECT DECOMPRESS ( Attachment, 'gzip' )
FROM TableA;

Since DECOMPRESS returns binary values, if the original values were of a character type, such as
LONG VARCHAR, a CAST can be applied to return human-readable values:

SELECT CAST ( DECOMPRESS ( Attachment, 'gzip' )
AS LONG VARCHAR ) FROM TableA;

**DECRIPT function [String]**

Decrypts the string using the supplied key and returns a LONG BINARY value.

**Syntax**

```
DECRIPT( string-expression, key
[, algorithm ]
)
```

- **algorithm** :
  - 'AES'
  - 'AES256'
  - 'AES_FIPS'
  - 'AES256_FIPS'
    [ format ]

- **format** :
  - ( FORMAT=RAW [,padding ] ) [ initialization-vector ]

- **padding** :
  - PADDING=PKCS5
  - ZEROES
  - NONE ]

**Parameters**

- **string-expression**  The string to be decrypted. Binary values can also be passed to this function. This parameter is case sensitive, even in case-insensitive databases.

- **key**  The encryption key (string) required to decrypt the string-expression. This value must be the same encryption key that was used to encrypt the string-expression to obtain the original value that was encrypted. This parameter is case sensitive, even in case-insensitive databases.

  **Caution**
  For strongly encrypted databases, store a copy of the key in a safe location. If you lose the encryption key, there is no way to access the data—even with the assistance of Technical Support. The database must be discarded and you must create a new database.

- **algorithm**  This optional parameter specifies the algorithm originally used to encrypt the string-expression.

  **FORMAT=RAW**  This optional parameter specifies that the data to be decrypted is in raw format. The initialization-vector parameter is required.
padding Specify the type of padding that was used to encrypt the data. If padding is not specified, PKCS5 is used by default.

The supported padding formats are:

PKCS5 The data is padded using the PKCS#5 algorithm. The decrypted data contains padding.

ZEROES The data is padded with zeros (0). The decrypted data is padded with zeros.

NONE The data is not padded. The decrypted data contains no padding.

initialization-vector Specify the initialization vector that was used to encrypt the data. This parameter is required.

Returns LONG BINARY

Remarks
You can use the DECRYPT function to decrypt a string-expression that was encrypted with the ENCRYPT function. This function returns a LONG BINARY value with the same number of bytes as the input string, unless the data is in raw format. When FORMAT=RAW, the length of the returned value depends on the padding format.

To successfully decrypt a string-expression, you must use the same encryption key that was used to encrypt the data. When FORMAT=RAW, you must also use the same initialization-vector and padding format that was used to encrypt the data. Data in raw format can be decrypted outside of the database server.

If you specify an incorrect encryption key, an error is generated unless FORMAT=RAW is specified. When you specify FORMAT=RAW and an incorrect encryption key or an incorrect initialization vector, the decryption fails silently.

Caution
For strongly encrypted data, store a copy of the key in a safe location. If you lose the encryption key, there is no way to access the data—even with the assistance of Technical Support.

Note
Not all platforms support FIPS-certified encryption. For a list of supported platforms, see http://www.sybase.com/detail?id=1061806.

See also
- “Raw encryption” [SQL Anywhere Server - Database Administration]
- “ENCRYPT function [String]” on page 226
- “ISENCRYPTED function [System]” on page 276
- “Column and table encryption” [SQL Anywhere Server - Database Administration]
- “String functions” on page 152
- “-fips database server option” [SQL Anywhere Server - Database Administration]
Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following example decrypts a user's password from the user_info table. The CAST function is used to convert the password back to a CHAR data type because the DECRYPT function converts values to the LONG BINARY data type, which is unreadable.

```sql
SELECT CAST( DECRYPT( user_pwd, '8U3dkA' ) AS CHAR(100) ) FROM user_info;
```

The following example decrypts data that was encrypted using the raw format. The data was encrypted with encryption key TheEncryptionKey and the initialization vector ThisIsTheIV.

```sql
SELECT DECRYPT( binary_data, 'TheEncryptionKey', 'AES(format=raw;padding=zeroes)', 'ThisIsTheIV'), LENGTH(binary_data) FROM SensitiveData;
```

### DEGREES function [Numeric]

Converts a number from radians to degrees.

#### Syntax

```
DEGREES( numeric-expression )
```

#### Parameters

- **numeric-expression**  An angle in radians.

#### Returns

DOUBLE

#### Remarks

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns the degrees of the angle given by `numeric-expression`. If the parameter is NULL, the result is NULL.

#### Standards and compatibility

- **SQL/2008**  Vendor extension.

#### Example

The following statement returns the value 29.79380534680281:

```sql
SELECT DEGREES( 0.52 );
```

### DENSE_RANK function [Ranking]

Calculates the rank of a value in a partition. For tied values, the DENSE_RANK function does not leave gaps in the ranking sequence.
Syntax

DENSE_RANK( ) OVER ( window-spec )

window-spec : see the Remarks section below

Returns

INTEGER

Remarks

Elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. When used as a window function, you must specify an ORDER BY clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE clause. More information is available in the window-spec definition of the WINDOW clause.

See also

- “WINDOW clause” on page 1051
- “CUME_DIST function [Ranking]” on page 200
- “PERCENT_RANK function [Ranking]” on page 316
- “RANK function [Ranking]” on page 326
- “Window functions” [SQL Anywhere Server - SQL Usage]
- “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 The DENSE_RANK function comprises part of optional SQL/2008 language feature T612, "Advanced OLAP operations".

SQL Anywhere supports SQL/2008 language feature F441, "Extended set function support", which permits operands of window functions to be arbitrary expressions that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the DENSE_RANK function, combined with an outer reference. For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example

The following example returns a result set that provides a ranking of the employees' salaries in Utah and New York. Although 19 records are returned in the result set, only 18 rankings are listed because of a 7th-place tie between the 7th and 8th employee in the list, who have identical salaries. Instead of ranking the 9th employee as '9', the employee is listed as '8' because the DENSE_RANK function does not leave gaps in the ranks.

```
SELECT DepartmentID, Surname, Salary, State, 
DENSE_RANK() OVER (ORDER BY Salary DESC) AS SalaryRank 
FROM GROUPO.Employees 
WHERE State IN ('NY','UT');
```

Here is the result set:
### DIFFERENCE function [String]

Returns the difference in the SOUNDEX values between the two string expressions.

**Syntax**

```
DIFFERENCE ( string-expression-1, string-expression-2 )
```
Parameters

- **string-expression-1**  The first SOUNDEX argument.
- **string-expression-2**  The second SOUNDEX argument.

Returns

SMALLINT

Remarks

The DIFFERENCE function compares the SOUNDEX values of two strings and evaluates the similarity between them, returning a value from 0 through 4, where 4 is the best match.

This function always returns some value. The result is NULL only if one of the arguments are NULL.

See also

- “SOUNDEX function [String]” on page 367
- “String functions” on page 152

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statement returns similarity between the words test and chest:

```sql
SELECT DIFFERENCE( 'test', 'chest' );
```

**DOW function [Date and time]**

Returns a number from 1 to 7 representing the day of the week of a date, where Sunday=1, Monday=2, and so on.

Syntax

```sql
DOW( date-expression )
```

Parameters

- **date-expression**  The value (of type DATE) to be evaluated.

Returns

SMALLINT

Remarks

The DOW function is not affected by the value specified for the first_day_of_week database option. For example, even if first_day_of_week is set to Monday, the DOW function returns a 2 for Monday.
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the value 5:

```sql
SELECT DOW( '1998-07-09' );
```

The following statement returns the value 1:

```sql
SELECT DOW( CAST( '2010/05/30 11:33:00.000000+04:00' as TIMESTAMP WITH TIME ZONE ) );
```

The following statement queries the Employees table and returns the employees StartDate, expressed as the number of the day of the week:

```sql
SELECT DOW( StartDate ) FROM GROUPO.Employees;
```

**ENCRYPT function [String]**

Encrypts the specified values using the supplied encryption key and returns a LONG BINARY value.

Syntax

```sql
ENCRYPT( string-expression, key
[, algorithm [ format ] ]
)
```

**algorithm** :

- 'AES'
- 'AES256'
- 'AES_FIPS'
- 'AES256_FIPS'

[ format ]

**format** :

( FORMAT=RAW [ ;padding ] [ initialization-vector ] )

**padding** :

PADDING=PKCS5

| ZEROES
| NONE ]

Parameters

**string-expression**  The data to be encrypted. Binary values can also be passed to this function. This parameter is case sensitive, even in case-insensitive databases.

**key**  The encryption key used to encrypt the string-expression. This same key must be used to decrypt the value to obtain the original value. This parameter is case sensitive, even in case-insensitive databases.

As with most passwords, it is best to choose a key value that cannot be easily guessed. It is recommended that you choose a value for your key that is at least 16 characters long, contains a mix of uppercase and
lowercase, and includes numbers, letters and special characters. You require this key each time you want to decrypt the data.

**Caution**
For strongly encrypted columns, store a copy of the key in a safe location. If you lose the encryption key, there is no way to access the data—even with the assistance of Technical Support. The column must be discarded and you must create a new column.

**algorithm** This optional parameter specifies the algorithm to use when encrypting *string-expression*. The algorithm used for strong encryption is Rijndael: a block encryption algorithm chosen as the new Advanced Encryption Standard (AES) for block ciphers by the National Institute of Standards and Technology (NIST).

You can specify one of the FIPS-certified algorithms for *algorithm* on any platform that supports FIPS-certified encryption.

**Note**
RSA and FIPS-certified encryption are not available on all platforms. For information about which platforms support which encryption method, see http://www.sybase.com/detail?id=1061806.

If *algorithm* is not specified, AES is used by default. If the database server was started using the -fips server option, AES_FIPS is used as the default instead.

**FORMAT=RAW** This optional parameter encrypts the data into raw format. The initialization vector parameter is required.

**padding** This optional parameter specifies the type of padding to use. If *padding* is not specified, PKCS5 is used by default.

**PKCS5** The data is padded using the PKCS#5 algorithm. The outputted (encrypted) data is 1-16 bytes longer than the input data.

**ZEROES** The data is padded with zeros (0). The outputted (encrypted) data is 0-15 bytes longer than the input data. When this output is decrypted, it is also padded with zeros.

**NONE** The data is not padded. The input data must be a multiple of the cipher block length (16-bytes).

**initialization-vector** This initialization vector parameter is required when FORMAT=RAW is specified. The string cannot be longer than 16 bytes. Any value less than 16 bytes is padded with 0 bytes. This string cannot be set to NULL.

**Returns**
LONG BINARY

**Remarks**
The LONG BINARY value returned by this function is at most 31 bytes longer than the input *string-expression*. The value returned by this function is not human-readable. You can use the DECRYPT function to decrypt a *string-expression* that was encrypted with the ENCRYPT function. To successfully
decrypt a string-expression, you must use the same encryption key and algorithm that were used to encrypt the data. If you specify an incorrect encryption key, an error is generated. A lost key results in inaccessible data, from which there is no recovery.

If you are storing encrypted values in a table, the column should be BINARY or LONG BINARY so that character set conversion is not performed on the data.

When FORMAT=RAW is specified, the data is encrypted using raw encryption. You must specify the encryption key, initialization vector, and optionally the padding format. These same values must be specified when decrypting the data. The decryption can be performed outside of the database server; although you can also use the DECRYPT function.

Raw encryption is not recommended when the data is to be encrypted and decrypted only within the database server because you must specify the initialization vector and the padding, and the encryption key cannot be verified during decryption.

See also
- “DECRYPT function [String]” on page 220
- “ISENCRYPTED function [System]” on page 276
- “Column and table encryption” [SQL Anywhere Server - Database Administration]
- “-fips database server option” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Example

The following trigger encrypts the user_pwd column of the user_info table. This column contains users' passwords, and the trigger fires whenever a password value is changed.

```sql
CREATE TRIGGER encrypt_updated_pwd
BEFORE UPDATE OF user_pwd
ON user_info
REFERENCING NEW AS new_pwd
FOR EACH ROW
BEGIN
    SET new_pwd.user_pwd=ENCRYPT( new_pwd.user_pwd, '8U3dkA' );
END;
```

The following SELECT statement uses raw encryption to encrypt the binary_data column of the SensitiveData table. The encrypted data is padded with zeroes:

```sql
SELECT ENCRYPT( binary_data, 'TheEncryptionKey', 'AES(format=raw;padding=zeroes)', 'ThisIsTheIV'), LENGTH(binary_data)
FROM SensitiveData;
```

**ERROR_LINE function [Miscellaneous]**

Returns the line number of the procedure or batch where the error occurred that invoked the CATCH block of a TRY...CATCH statement.
Syntax
ERROR_LINE()

Returns
UNSIGNED INTEGER representing the line number within the stored procedure or the compound statement where an error occurred.

Remarks
Call this function anywhere within a CATCH block. This function reports information about the current error when it is invoked within an error handler, a nested compound statement, a function, or a procedure.

See also
- “TRY statement” on page 1022
- “BEGIN statement” on page 523
- “ERROR_MESSAGE function [Miscellaneous]” on page 229
- “ERROR_PROCEDURE function [function type]” on page 230
- “ERROR_SQLCODE function [Miscellaneous]” on page 231
- “ERROR_SQLSTATE function [Miscellaneous]” on page 232
- “ERROR_STACK_TRACE function [Miscellaneous]” on page 233
- “STACK_TRACE function [Miscellaneous]” on page 371
- “sa_error_stack_trace system procedure” on page 1136
- “sa_stack_trace system procedure” on page 1250
- “Nested compound statements and exception handlers” [SQL Anywhere Server - SQL Usage]
- “Example: Creating an error logging procedure that can be called by an exception handler” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
When executed within a handler that was invoked by a division by zero error on line 15 of the procedure u1.proc1, the following statement SELECT ERROR_LINE( ), ERROR_MESSAGE( ), ERROR_PROCEDURE( ) returns a result similar to the following one:

15, 'Division by zero', 'u1'."proc1"

ERROR_MESSAGE function [Miscellaneous]
Returns the message text of the error that invoked the CATCH block of a TRY...CATCH statement.

Syntax
ERROR_MESSAGE( )

Returns
VARCHAR containing the error message of the error that invoked the CATCH block.
Remarks

Call this function anywhere within a CATCH block. This function returns the active error message anywhere in the error handler, while the ERRORMSG function, when called with no parameters, only returns the error message when invoked in the first statement of the error handler.

The parameters in the error message are replaced with actual values.

See also

- “TRY statement” on page 1022
- “BEGIN statement” on page 523
- “ERROR_LINE function [Miscellaneous]” on page 228
- “ERRORPROCEDURE function [function type]” on page 230
- “ERRORSQLCODE function [Miscellaneous]” on page 231
- “ERRORSQLSTATE function [Miscellaneous]” on page 232
- “ERRORSTACKTRACE function [Miscellaneous]” on page 233
- “STACKTRACE function [Miscellaneous]” on page 371
- “sa_error_stack_trace system procedure” on page 1136
- “sa_stack_trace system procedure” on page 1250
- “Nested compound statements and exception handlers” [SQL Anywhere Server - SQL Usage]
- “Example: Creating an error logging procedure that can be called by an exception handler” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

When executed within a handler that was invoked by a division by zero error on line 15 of the procedure u1.proc1, the following statement

```
SELECT ERROR_LINE(), ERROR_MESSAGE(),
ERRORPROCEDURE()
```

returns the following result:

```
15, 'Division by zero', '"u1"."proc1"
```

**ERRORPROCEDURE function [function type]**

Returns the name of the procedure within which the error that caused the exception handler to run occurred.

**Syntax**

```
ERRORPROCEDURE()
```

**Returns**

VARCHAR containing the qualified name of the procedure where the error has occurred when called in a TRY..CATCH statement. If the compound statement is not part of a procedure, function, trigger, or event, the type of batch (watcom_batch or tsql_batch) is returned instead of the procedure name.
Remarks

ERROR_PROCEDURE can be called anywhere within a TRY...CATCH block.

See also

- “TRY statement” on page 1022
- “BEGIN statement” on page 523
- “ERROR_LINE function [Miscellaneous]” on page 228
- “ERROR_MESSAGE function [Miscellaneous]” on page 229
- “ERROR_SQLCODE function [Miscellaneous]” on page 231
- “ERROR_SQLSTATE function [Miscellaneous]” on page 232
- “ERROR_STACK_TRACE function [Miscellaneous]” on page 233
- “STACK_TRACE function [Miscellaneous]” on page 371
- “sa_error_stack_trace system procedure” on page 1136
- “Nested compound statements and exception handlers” [SQL Anywhere Server - SQL Usage]
- “Example: Creating an error logging procedure that can be called by an exception handler” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

Consider the following statement:

```sql
SELECT ERROR_LINE(), ERROR_MESSAGE(), ERROR_PROCEDURE();
```

When this statement is executed within a handler that was invoked by a division by zero error, it returns the following result:

```
15, 'Division by zero', '"u1"."proc1"
```

The statement, `SELECT ERROR_PROCEDURE();`, when executed by a handler that was invoked by a division by zero error in the procedure u1.proc1, returns the following result:

```
"u1"."proc1"
```

**ERROR_SQLCODE function [Miscellaneous]**

Returns the SQLCODE of the error that invoked the error handler.

Syntax

```sql
ERROR_SQLCODE()
```

Returns

SIGNED INTEGER with the value of the SQLCODE of the error that invoked the error handler.

Remarks

This function can be called anywhere within a TRY statement, and it returns the SQLCODE anywhere in the error handler.
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The statement `SELECT ERROR_SQLSTATE( ), ERROR_SQLCODE( );`, when executed within a handler that was invoked by a division by zero error, returns the following result:

'22012', -628

**ERROR_SQLSTATE function [Miscellaneous]**

Returns the SQLSTATE of the error that invoked the error handler.

Syntax

```
ERROR_SQLSTATE()
```

Returns

CHAR(5) representing the SQLSTATE of the error that invoked the error handler.

Remarks

This function can be called anywhere within a TRY statement, and it returns the SQLSTATE anywhere in the error handler.
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The statement `SELECT ERROR_SQLSTATE( ), ERROR_SQLCODE( );`, when executed within a handler that was invoked by a division by zero error, returns the following result:

'22012', -628

**ERROR_STACK_TRACE function [Miscellaneous]**

Returns the information about the stack trace of the error that invoked the error handler.

**Syntax**

`ERROR_STACK_TRACE()`

**Returns**

LONG VARCHAR representing the stack trace of the error that invoked the error handler. If the compound statement is not part of a procedure, function, trigger, or event, the type of batch (watcom_batch or tsql_batch) is returned instead of the procedure name.

**Remarks**

Each line of the returned value contains the qualified procedure name or batch type (if any) of the statement on the stack, followed by the line number of the statement. The last line of the returned value does not end in the new line character. If a procedure is hidden, the database server does not output the procedure name that invoked the error.

This function returns the same information as the sa_error_stack_trace system procedure.
Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following set of procedures (with line numbers added) can be used to obtain the error stack:

```sql
1 CREATE PROCEDURE proc1()
2 BEGIN TRY
3    CALL proc2();
4 END TRY
5 BEGIN CATCH
6    SELECT ERROR_STACK_TRACE();
7 END CATCH;

CREATE PROCEDURE proc2()
2 BEGIN
3    CALL proc3();
4 END;

CREATE PROCEDURE proc3()
2 BEGIN
3    DECLARE v INTEGER = 0;
4    SET v = 1 / v;
5 END;

CALL proc1();
```

This call returns the following result string:

```
"DBA"."proc1" : 3
"DBA"."proc2" : 3
"DBA"."proc3" : 4
```

If RESIGNAL is used in the error handler, and the resignaled error is handled, the error stack reported in the second handler contains the stack trace of the original error, the record of the RESIGNAL, and the stack of the resignaled exception. For example:

```sql
CREATE PROCEDURE proc1()
BEGIN TRY
    BEGIN TRY
        CALL proc1();
    END TRY
```
DECLARE v INTEGER = 0;
SET v = 1 / v;
END TRY
BEGIN CATCH
    CALL proc2();
END CATCH
END TRY
BEGIN CATCH
    SELECT ERROR_STACK_TRACE();
END CATCH;

CREATE PROCEDURE proc2()
BEGIN
    CALL proc3();
END;

CREATE PROCEDURE proc3()
BEGIN
    RESIGNAL;
END;

CALL proc1();

This call returns the following result string:

"DBA"."proc1" : 8
"DBA"."proc2" : 3
RESIGNAL: "DBA"."proc3" : 3
"DBA"."proc1" : 5

ERRORMSG function [Miscellaneous]

Provides the error message for the current error, or for a specified SQLSTATE or SQLCODE value.

Syntax

ERRORMSG([sqlstate | sqlcode])

Parameters

- **sqlstate**: string
- **sqlcode**: integer

Returns

VARCHAR containing the error message.

Remarks

If no argument is supplied, the error message for the current state is supplied. Any substitutions (such as table names and column names) are made.
If an argument is supplied, the error message for the supplied SQLSTATE or SQLCODE is returned, with no substitutions. Table names and column names are supplied as placeholders (%1).

See also
- “SQL Anywhere error messages sorted by SQLSTATE” [Error Messages]
- “SQL Anywhere error messages sorted by SQLCODE” [Error Messages]

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement returns the error message for SQLCODE -813:

```sql
SELECT ERRORMSG( -813 );
```

**ESTIMATE function [Miscellaneous]**

Returns selectivity estimates as a percentage calculated by the query optimizer, based on specified parameters.

**Syntax**

```sql
ESTIMATE( column-name [, value [, relation-string ]] )
```

**Parameters**
- `column-name`  The column used in the estimate.
- `value`  The value to which the column is compared. The default is NULL.
- `relation-string`  The comparison operator used for the comparison, enclosed in single quotes. Possible values for this parameter are: '=' , '>' , '<' , '>=', '<=', '<>', '!=' , '!=' , '<' , and '>'. The default is '='.

**Returns**

REAL

**Remarks**

This function returns selectivity estimates for the predicate `column-name relation-string value`. If `value` is NULL and the relation string is '=' , the selectivity is for the predicate `column-name IS NULL`. If `value` is NULL and the relation string is '!=' or '<>' , the selectivity is for the predicate `column-name IS NOT NULL`.
See also

- “Selectivity estimate sources” [SQL Anywhere Server - SQL Usage]
- “Selectivity information in the graphical plan” [SQL Anywhere Server - SQL Usage]
- “INDEX_ESTIMATE function [Miscellaneous]” on page 273
- “ESTIMATE_SOURCE function [Miscellaneous]” on page 237
- “EXPERIENCE_ESTIMATE function [Miscellaneous]” on page 243

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statement returns the percentage of EmployeeID values estimated to be greater than 200. The precise value depends on the actions you have carried out on the database.

```sql
SELECT FIRST ESTIMATE( EmployeeID, 200, '>' )
FROM GROUPO.Employees
ORDER BY 1;
```

**ESTIMATE_SOURCE function [Miscellaneous]**

Provides the source for selectivity estimates used by the query optimizer.

Syntax

```sql
ESTIMATE_SOURCE(
    column-name
    [, value
    [, relation-string ] ]
)
```

Parameters

- **column-name**  The name of the column that is being investigated.
- **value**  The value to which the column is compared. The default is NULL.
- **relation-string**  The comparison operator used for the comparison, enclosed in single quotes. Possible values for this parameter are: '=' , '>' , '<' , '>=', '<=', '<>', '!=', '!=<', and '!=>'. The default is '='.

Returns

The following list shows the selectivity estimate sources that ESTIMATE_SOURCE returns. For more information about the sources, see “Selectivity estimate sources” [SQL Anywhere Server - SQL Usage].

<table>
<thead>
<tr>
<th>Value</th>
<th>Selectivity estimate source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistics</td>
<td>Stored column statistics</td>
</tr>
<tr>
<td>Column</td>
<td>Average of all values stored in the column statistics</td>
</tr>
<tr>
<td>Value</td>
<td>Selectivity estimate source</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Index</td>
<td>Index probes</td>
</tr>
<tr>
<td>Guess</td>
<td>Built-in guesses that are defined for each type of predicate. This is returned only when there is no relevant index to use, no statistics have been collected for the referenced columns, or the predicate is a complex predicate.</td>
</tr>
<tr>
<td>Computed</td>
<td>Other sources than the ones described above</td>
</tr>
<tr>
<td>Always</td>
<td>Returned when the specified predicate is always true</td>
</tr>
<tr>
<td>Combined</td>
<td>One or more of the above sources</td>
</tr>
<tr>
<td>Bounded</td>
<td>Returned when there are upper and/or lower bounds placed on the selectivity estimate</td>
</tr>
</tbody>
</table>

**Remarks**

This function returns the source of the selectivity estimate for the predicate `column-name relation-string value`. If `value` is NULL and the relation string is `='`, the selectivity source is for the predicate `column-name IS NULL`. If `value` is NULL and the relation string is `!='` or `<>`, the selectivity source is for the predicate `column-name IS NOT NULL`.

**See also**

- “Selectivity estimate sources” [SQL Anywhere Server - SQL Usage]
- “ESTIMATE function [Miscellaneous]” on page 236
- “INDEX_ESTIMATE function [Miscellaneous]” on page 273

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statement returns the selectivity source Index for evaluating whether the first value in the EmployeeID column is greater than 200. Returning Index means that the query optimizer used an index to estimate the selectivity.

```sql
SELECT FIRST ESTIMATE_SOURCE( EmployeeID, 200, '>' )
FROM GROUPO.Employees
ORDER BY 1;
```

**EVENT_CONDITION function [System]**

Specifies when an event handler is triggered.

**Syntax**

```sql
EVENT_CONDITION( condition-name )
```
Parameters

- **condition-name**  The condition triggering the event. The possible values are preset in the database, and are case insensitive. Each condition is valid only for certain event types. The conditions and the events for which they are valid are as follows:

<table>
<thead>
<tr>
<th>Condition name</th>
<th>Units</th>
<th>Valid for...</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBFreePercent</td>
<td>n/a</td>
<td>DBDiskSpace</td>
<td></td>
</tr>
<tr>
<td>DBFreeSpace</td>
<td>MB</td>
<td>DBDiskSpace</td>
<td></td>
</tr>
<tr>
<td>DBSize</td>
<td>MB</td>
<td>GrowDB</td>
<td></td>
</tr>
<tr>
<td>ErrorNumber</td>
<td>n/a</td>
<td>RAISERROR</td>
<td></td>
</tr>
<tr>
<td>IdleTime</td>
<td>seconds</td>
<td>ServerIdle</td>
<td></td>
</tr>
<tr>
<td>Interval</td>
<td>seconds</td>
<td>All</td>
<td>Time since handler last executed</td>
</tr>
<tr>
<td>LogFreePercent</td>
<td>n/a</td>
<td>LogDiskSpace</td>
<td></td>
</tr>
<tr>
<td>LogFreeSpace</td>
<td>MB</td>
<td>LogDiskSpace</td>
<td></td>
</tr>
<tr>
<td>LogSize</td>
<td>MB</td>
<td>GrowLog</td>
<td></td>
</tr>
<tr>
<td>RemainingValues</td>
<td>integer</td>
<td>GlobalAutoincrement</td>
<td>The number of remaining values</td>
</tr>
<tr>
<td>TempFreePercent</td>
<td>n/a</td>
<td>TempDiskSpace</td>
<td></td>
</tr>
<tr>
<td>TempFreeSpace</td>
<td>MB</td>
<td>TempDiskSpace</td>
<td></td>
</tr>
<tr>
<td>TempSize</td>
<td>MB</td>
<td>GrowTemp</td>
<td></td>
</tr>
</tbody>
</table>

Returns

INT

Remarks

The EVENT_CONDITION function returns NULL when not called from an event.

See also

- “CREATE EVENT statement” on page 570

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following event definition uses the EVENT_CONDITION function:
CREATE EVENT LogNotifier
    TYPE LogDiskSpace
    WHERE event_condition( 'LogFreePercent' ) < 50
    HANDLER
      BEGIN
        MESSAGE 'LogNotifier message'
      END;

**EVENT_CONDITION_NAME function [System]**

Lists the possible parameters for EVENT_CONDITION.

**Syntax**

```
EVENT_CONDITION_NAME( integer )
```

**Parameters**

- `integer` Must be greater than or equal to zero.

**Returns**

VARCHAR

**Remarks**

You can use the EVENT_CONDITION_NAME function to obtain a list of all arguments for the EVENT_CONDITION function by looping over integers until the function returns NULL.

The EVENT_CONDITION_NAME function returns NULL when not called from an event.

**See also**

- “CREATE EVENT statement” on page 570

**Standards and compatibility**

- SQL/2008 Vendor extension.

**EVENT_PARAMETER function [System]**

Provides context information for event handlers.

**Syntax**

```
EVENT_PARAMETER( context-name )
```

`context-name`:
- AppInfo
- ConnectionID
- DisconnectReason
- EventName
- Executions
- MirrorServerName
- NumActive
Parameters

- **context-name**  One of the preset strings. The strings must be quoted, are case insensitive, and carry the following information:
  
  - **AppInfo**  The value of the AppInfo connection property for the connection that caused the event to be triggered. Use the following statement to see the value of the property outside the context of the event:
    
    ```sql
    SELECT CONNECTION_PROPERTY( 'AppInfo' );
    ```
    
    The AppInfo string contains the computer name and application name of the client connection for embedded SQL, ODBC, OLE DB, ADO.NET, and SQL Anywhere JDBC driver connections.
  
  - **ConnectionID**  The connection ID of the connection that caused the event to be triggered.
  
  - **DisconnectReason**  A string indicating the reason the connect was terminated. This parameter is valid only for Disconnect events. Possible results include:
    
    - **abnormal**  A disconnect occurred as a result of the client application terminating abnormally before disconnecting from the database, or as a result of a communication failure between the client and server computers.
    
    - **connect failed**  A connection attempt failed.
    
    - **drop connection**  A DROP CONNECTION statement was executed.
    
    - **from client**  The client application disconnected.
    
    - **inactive**  No requests were received for the period specified by the -ti server option.
    
    - **liveness**  No liveness packets were received for the period specified by the -tl server option.
  
  - **EventName**  The name of the event that has been triggered.
  
  - **Executions**  The number of times the event handler has been executed.
  
  - **MirrorServerName**  The name of the mirror or arbiter server that lost its connection to the primary server in a database mirroring system.
  
  - **NumActive**  The number of active instances of an event handler. This is useful for limiting an event handler so that only one instance executes at any given time.
  
  - **ScheduleName**  The name of the schedule which caused an event to be fired. If the event was fired manually using TRIGGER EVENT or as a system event, the result will be an empty string. If the schedule was not assigned a name explicitly when it was created, its name is the name of the event.
○ **SQLCODE**  The SQLCODE of the error that occurred during a failed connection. This parameter is valid only for ConnectFailed events.

○ **TableName**  The name of the table, for use with RemainingValues.

○ **User**  The user ID for the user that caused the event to be triggered.

In addition, you can access any of the valid condition-name arguments to the EVENT_CONDITION function from the EVENT_PARAMETER function.

The following table indicates which context-name values are valid for which system event types.

<table>
<thead>
<tr>
<th>System event types</th>
<th>Context-name value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackupEnd</td>
<td>AppInfo, ConnectionID, EventName, Executions, NumActive, User</td>
</tr>
<tr>
<td>Connect</td>
<td>AppInfo, ConnectionID, EventName, Executions, NumActive, User</td>
</tr>
<tr>
<td>ConnectFailed</td>
<td>AppInfo, EventName, Executions, NumActive, SQLCODE, User</td>
</tr>
<tr>
<td>&quot;Disconnect&quot;</td>
<td>AppInfo, ConnectionID, EventName, Executions, NumActive, User</td>
</tr>
<tr>
<td>GlobalAutoincrement</td>
<td>ConnectionID, EventName, Executions, NumActive, TableName, User</td>
</tr>
<tr>
<td>&quot;RAISERROR&quot;</td>
<td>AppInfo, ConnectionID, EventName, Executions, NumActive, User</td>
</tr>
<tr>
<td>User events</td>
<td>AppInfo, ConnectionID, EventName, Executions, NumActive, User</td>
</tr>
</tbody>
</table>

**Returns**

VARCHAR

**Remarks**

The maximum size of values passed to an event is limited by the maximum page size for the server (-gp server option). Values that are longer are truncated to be less than the maximum page size.

**See also**

- “EVENT_CONDITION function [System]” on page 238
- “CREATE EVENT statement” on page 570
- “TRIGGER EVENT statement” on page 1018
- “-gp database server option” [SQL Anywhere Server - Database Administration]

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

**Example**

The following example shows how to pass a string parameter to an event. The event displays the time it was triggered in the database server messages window.
CREATE EVENT ev_PassedParameter
    HANDLER
    BEGIN
        MESSAGE 'ev_PassedParameter - was triggered at ' ||
        event_parameter( 'time' );
    END;
    TRIGGER EVENT ev_PassedParameter( "Time"=string(current timestamp ) );

**EXP function [Numeric]**

Returns the result of the base of natural logarithms e raised to the power of the given argument.

**Syntax**

```
EXP( numeric-expression )
```

**Parameters**

- `numeric-expression` The exponent.

**Returns**

DOUBLE

**Remarks**

The EXP function returns the result of raising the base of natural logarithms e by the value specified by `numeric-expression`.

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

**Standards and compatibility**

- **SQL/2008** The EXP function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

**Example**

The statement returns the value 3269017.3724721107:

```
SELECT EXP( 15 );
```

**EXPERIENCE_ESTIMATE function [Miscellaneous]**

Returns selectivity estimates as a percentage calculated by the query optimizer, based on specified parameters.

**Syntax**

```
EXPERIENCE_ESTIMATE(
    column-name
    [, value
    [, relation-string ] ]
)
```
Parameters

- **column-name**  The name of the column that is being investigated.
- **value**  The value to which the column is compared.
- **relation-string**  The comparison operator used for the comparison. Possible values for this parameter are: '=' , '>' , '<' , '>=', '<=', '>', '!=', '!=', '<', '>', and '!=>'. The default is '='.

Returns

REAL

Remarks

If value is NULL then the relation strings = and != are interpreted as the IS NULL and IS NOT NULL conditions, respectively.

See also

- “ESTIMATE function [Miscellaneous]” on page 236
- “INDEX_ESTIMATE function [Miscellaneous]” on page 273
- “ESTIMATE_SOURCE function [Miscellaneous]” on page 237

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns 90.3262405396:

```
SELECT DISTINCT EXPERIENCE_ESTIMATE( EmployeeID, 200, '>' )
FROM GROUPO.Employees;
```

EXPLANATION function [Miscellaneous]

Returns the optimization strategy of a SQL statement as a plain text string.

Syntax

```
EXPLANATION(
    string-expression
    [, cursor-type ]
    [, update-status ]
)
```

Parameters

- **string-expression**  The SQL statement, which is commonly a SELECT statement, but can also be an UPDATE, MERGE, or DELETE statement.
- **cursor-type**  A cursor type, expressed as a string. Possible values are asensitive, insensitive, sensitive, or keyset-driven. If cursor-type is not specified, asensitive is used by default.
- **update-status** A string parameter accepting one of the following values indicating how the optimizer should treat the given cursor:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ-ONLY</td>
<td>The cursor is read-only.</td>
</tr>
<tr>
<td>READ-WRITE (default)</td>
<td>The cursor can be read or written to.</td>
</tr>
<tr>
<td>FOR UPDATE</td>
<td>The cursor can be read or written to. This is the same as READ-WRITE.</td>
</tr>
</tbody>
</table>

**Returns**

LONG VARCHAR

**Remarks**

The statement’s access plan is returned as a string. For information about interpreting the result, see “Advanced: Query execution plans” [SQL Anywhere Server - SQL Usage].

The GRAPHICAL_PLAN function offers significantly greater information about access plans, including system properties that may have affected how the statement was optimized.

This information can help you decide which indexes to add or how to structure your database for better performance.

**See also**

- “Execution plans in UltraLite” [UltraLite - Database Management and Reference]
- “Advanced: Query execution plans” [SQL Anywhere Server - SQL Usage]
- “PLAN function [Miscellaneous]” on page 318
- “GRAPHICAL_PLAN function [Miscellaneous]” on page 252

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statement passes a SELECT statement as a string parameter and returns the plan for executing the query:

```
SELECT EXPLANATION( 'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

The following statement returns a string containing the short form of the text plan for an INSENSITIVE cursor over the query 'select * from Departments where ....':

```
SELECT EXPLANATION( 'SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100', 'insensitive', 'read-only' );
```
EXPRTYPE function [Miscellaneous]

Returns a string that identifies the data type of an expression.

Syntax

```
EXPRTYPE( string-expression, integer-expression )
```

Parameters

- `string-expression`  A SELECT statement. The expression whose data type is to be queried must appear in the SELECT list. If the string is not a valid SELECT statement, NULL is returned.

- `integer-expression`  The position in the SELECT list of the desired expression. The first item in the SELECT list is numbered 1. If the integer-expression value does not correspond to a SELECT list item, NULL is returned.

Returns

LONG VARCHAR

Remarks

For user-defined domains, EXPRTYPE returns the description of the underlying data type, not the domain name. For example, suppose you create a domain, mydomain, and define a table column using mydomain, as follows:

```
CREATE DOMAIN mydomain CHAR(20);
CREATE TABLE mytable( colA mydomain, colB DATETIME );
```

When you execute `SELECT EXPRTYPE( 'SELECT * FROM mytable', 1 )`, the data type returned is char(20), not mydomain.

See also

- “SQL data types” on page 89
- “sa_describe_query system procedure” on page 1127

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns `smallint` when executed against the SQL Anywhere sample database:

```
SELECT EXPRTYPE( 'SELECT LineID FROM SalesOrderItems', 1 );
```

FIRST_VALUE function [Aggregate]

Returns values from the first row of a window.
Syntax

FIRST_VALUE([ ALL ] expression [ { RESPECT | IGNORE } NULLS ])
OVER ( window-spec )

window-spec : see the Remarks section below

Parameters

- **expression**  The expression to evaluate. For example, a column name.

Returns

Data type of the values from the first row of a window.

Remarks

The FIRST_VALUE function allows you to select the first value (according to some ordering) in a table, without having to use a self-join. This is valuable when you want to use the first value as the baseline in calculations.

The FIRST_VALUE function takes the first record from the window. Then, the *expression* is computed against the first record and results are returned.

If IGNORE NULLS is specified, the first non-NULL value of *expression* is returned. If RESPECT NULLS is specified (the default), the first value is returned whether or not it is NULL.

The FIRST_VALUE function is different from most other aggregate functions in that it can only be used with a window specification.

Elements of *window-spec* can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. More information is provided in the *window-spec* definition of the WINDOW clause. See “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also

- “Window aggregate functions” [SQL Anywhere Server - SQL Usage]
- “LAST_VALUE function [Aggregate]” on page 279

Standards and compatibility

- **SQL/2008**  Vendor extension.

SQL Anywhere supports SQL/2008 language feature F441, "Extended set function support", which permits operands of window functions to be arbitrary expressions that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a
column reference from the query block containing the FIRST_VALUE function, combined with an outer reference. For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example
The following example returns the relationship, as a percentage, between each employee's salary and that of the most recently hired employee in the same department:

```
SELECT DepartmentID, EmployeeID,
       100 * Salary / ( FIRST_VALUE( Salary ) OVER ( PARTITION BY DepartmentID ORDER BY StartDate DESC ) )
       AS percentage
FROM GROUPO.Employees;
```

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>EmployeeID</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1658</td>
<td>100</td>
</tr>
<tr>
<td>500</td>
<td>1615</td>
<td>110.4284624</td>
</tr>
<tr>
<td>500</td>
<td>1570</td>
<td>138.8427097</td>
</tr>
<tr>
<td>500</td>
<td>1013</td>
<td>109.5851905</td>
</tr>
<tr>
<td>500</td>
<td>921</td>
<td>167.4497049</td>
</tr>
<tr>
<td>500</td>
<td>868</td>
<td>113.2393688</td>
</tr>
<tr>
<td>500</td>
<td>750</td>
<td>137.7344095</td>
</tr>
<tr>
<td>500</td>
<td>703</td>
<td>222.8679276</td>
</tr>
<tr>
<td>500</td>
<td>191</td>
<td>119.6642975</td>
</tr>
<tr>
<td>400</td>
<td>1751</td>
<td>100</td>
</tr>
<tr>
<td>400</td>
<td>1740</td>
<td>99.705647</td>
</tr>
<tr>
<td>400</td>
<td>1684</td>
<td>130.969936</td>
</tr>
<tr>
<td>400</td>
<td>1643</td>
<td>83.9734797</td>
</tr>
<tr>
<td>400</td>
<td>1607</td>
<td>175.1828989</td>
</tr>
<tr>
<td>400</td>
<td>1576</td>
<td>197.0164609</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Employee 1658 is the first row for department 500, indicating that they are the most recent hire in that department and their percentage is 100%. Percentages for the remaining department 500 employees are...
calculated relative to that of employee 1658. For example, employee 1570 earns approximately 139\% of what employee 1658 earns.

If another employee in the same department makes the same salary as the most recent hire, they will have a percentage of 100 as well.

**FLOOR function [Numeric]**

Returns the largest integer not greater than the given number.

**Syntax**

\[ \text{FLOOR}( \text{numeric-expression} ) \]

**Parameters**

- **numeric-expression** The value to be truncated, typically a fixed numeric type with non-zero scale or an approximate numeric type (DOUBLE, REAL, or FLOAT).

**Returns**

DOUBLE

**Remarks**

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

**See also**

- “CEILING function [Numeric]” on page 175

**Standards and compatibility**

- SQL/2008 The FLOOR function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

**Example**

The following statement returns a Floor value of 123:

\[ \text{SELECT FLOOR (123)}; \]

The following statement returns a Floor value of 123:

\[ \text{SELECT FLOOR (123.45)}; \]

The following statement returns a Floor value of -124:

\[ \text{SELECT FLOOR (-123.45)}; \]

**GET_BIT function [Bit array]**

Returns the value (1 or 0) of a specified bit in a bit array.
Syntax

```
GET_BIT( bit-expression, position )
```

Parameters
- **bit-expression**  The bit array containing the bit.
- **position**  The position of the bit for which to return the status.

Returns

```
BIT
```

Remarks

The positions in the array are counted from the left side, starting at 1.

If `position` exceeds the length of the array, 0 (false) is returned.

See also

- “Bitwise operators” on page 19
- “SET_BIT function [Bit array]” on page 359
- “SET_BITS function [Aggregate]” on page 360
- “sa_get_bits system procedure” on page 1141

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statement returns the value 1:

```
SELECT GET_BIT( '00110011', 4 );
```

The following statement returns the value 0:

```
SELECT GET_BIT( '00110011', 5 );
```

**GET_IDENTITY function [Miscellaneous]**

Allocates values to an AUTOINCREMENT column. This is an alternative to using AUTOINCREMENT to generate numbers.

Syntax

```
GET_IDENTITY( table_name [, number_to_allocate ] )
```

Parameters
- **table_name**  A string indicating the name of the table, including, optionally, the owner name.
- **number_to_allocate**  The number of values to reserve. The default is 1.
Returns
UNSIGNED BIGINT

Remarks
Using AUTOINCREMENT or GLOBAL AUTOINCREMENT is still the most efficient way to generate IDs, but this function is provided as an alternative. The function assumes that the table has an AUTOINCREMENT column defined. It returns the next available value that would be generated for the table's AUTOINCREMENT column, and reserves that value so that no other connection will use it by default.

The function returns an error if the table is not found, and returns NULL if the table has no AUTOINCREMENT column. If there is more than one AUTOINCREMENT column, it uses the first one it finds.

`number_to_allocate` is the number of values to reserve. If `number_to_allocate` is greater than 1, the function also reserves the remaining values. The next allocation uses the current number plus the value of `number_to_allocate`. This allows the application to execute the GET_IDENTITY function less frequently. If `number_to_allocate` is 0, the next available value is returned without reserving any values.

No COMMIT is required after executing the GET_IDENTITY function, and so it can be called using the same connection that is used to insert rows. If ID values are required for several tables, they can be obtained using a single SELECT that includes multiple calls to the GET_IDENTITY function, as in the example.

The GET_IDENTITY function is non-deterministic function; successive calls to it may return different values. The optimizer does not cache the results of the GET_IDENTITY function.

For more information about non-deterministic functions, see “Function caching” [SQL Anywhere Server - SQL Usage].

See also
● “CREATE TABLE statement” on page 690
● “ALTER TABLE statement” on page 486
● “NUMBER function [Miscellaneous]” on page 313

Standards and compatibility
● SQL/2008 Vendor extension.

Example
The following statement returns the next available value for the Customers table AUTOINCREMENT column (ID). The number returned and the following nine values are reserved:

```sql
SELECT GET_IDENTITY( 'GROUPO.Customers', 10 );
```

GETDATE function [Date and time]
Returns the current year, month, day, hour, minute, second, and fraction of a second.
Syntax

GETDATE()

Returns

TIMESTAMP

Remarks

The accuracy is limited by the accuracy of the system clock.

The information the GETDATE function returns is equivalent to the information returned by the NOW function and the CURRENT TIMESTAMP special value.

See also

- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMP special value” on page 70
- “DATE data type” on page 115
- “DATE function [Date and time]” on page 203
- “DATETIME data type” on page 116
- “DATETIME function [Date and time]” on page 209
- “DATETIMEOFFSET data type” on page 117
- “Expressions” on page 21
- “ISDATE function [Data type conversion]” on page 275
- “NOW function [Date and time]” on page 311
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP special value” on page 76
- “TIMESTAMP data type” on page 121
- “UTC TIMESTAMP special value” on page 78

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns the system date and time:

SELECT GETDATE();

GRAPHICAL_PLAN function [Miscellaneous]

Returns the plan optimization strategy of a SQL statement in XML format, as a string.

Syntax

GRAPHICAL_PLAN(
    string-expression
[, statistics-level]
Parameters

- **string-expression**  The SQL statement, which is commonly a SELECT statement but which may also be an UPDATE or DELETE statement.

- **statistics-level**  An integer. Statistics-level can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Optimizer estimates only (default).</td>
</tr>
<tr>
<td>2</td>
<td>Detailed statistics including node statistics.</td>
</tr>
<tr>
<td>3</td>
<td>Detailed statistics.</td>
</tr>
</tbody>
</table>

- **cursor-type**  A cursor type, expressed as a string. Possible values are: asensitive, insensitive, sensitive, or keyset-driven. If cursor-type is not specified, asensitive is used by default.

- **update-status**  A string parameter accepting one of the following values indicating how the optimizer should treat the given cursor:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ-ONLY</td>
<td>The cursor is read-only.</td>
</tr>
<tr>
<td>READ-WRITE (default)</td>
<td>The cursor can be read or written to.</td>
</tr>
<tr>
<td>FOR UPDATE</td>
<td>The cursor can be read or written to. This is exactly the same as READ-WRITE.</td>
</tr>
</tbody>
</table>

Returns

LONG VARCHAR

See also

- “Advanced: Query execution plans” [SQL Anywhere Server - SQL Usage]
- “PLAN function [Miscellaneous]” on page 318
- “EXPLANATION function [Miscellaneous]” on page 244

Standards and compatibility

- SQL/2008  Vendor extension.

Examples

The following Interactive SQL example passes a SELECT statement as a string parameter and returns the plan for executing the query. It saves the plan in the file `plan.saplan` which can be opened and read using Interactive SQL.
SELECT GRAPHICAL_PLAN( 'SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100' );
OUTPUT TO 'plan.saplan' FORMAT TEXT QUOTE '' HEXADECIMAL ASIS;

The following statement returns a string containing the graphical plan for a keyset-driven, updatable cursor over the query `SELECT * FROM Departments WHERE GROUPO.DepartmentID > 100`. It also causes the server to annotate the plan with actual execution statistics, in addition to the estimated statistics that were used by the optimizer.

SELECT GRAPHICAL_PLAN( 'SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100', 2, 'keyset-driven', 'for update' );

**GREATER function [Miscellaneous]**

Returns the greater of two parameter values.

**Syntax**

```
GREATER( expression-1, expression-2 )
```

**Parameters**

- **expression-1**  The first parameter value to be compared.
- **expression-2**  The second parameter value to be compared.

**Returns**

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

**Remarks**

If the parameters are equal, the first is returned.

**See also**

- “LESSER function [Miscellaneous]” on page 284

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

The following statement returns the value 10:

```
SELECT GREATER( 10, 5 ) FROM dummy;
```
GROUPING function [Aggregate]

Identifies whether a column in a GROUP BY operation result set is NULL because it is part of a subtotal row, or NULL because of the underlying data.

**Syntax**

GROUPING( group-by-expression )

**Parameters**

- **group-by-expression**  An expression appearing as a grouping column in the result set of a query that uses a GROUP BY clause. This function can be used to identify subtotal rows added to the result set by a ROLLUP or CUBE operation.

**Returns**

- **1**  Indicates that group-by-expression is NULL because it is part of a subtotal row. The column is not a prefix column for that row.

- **0**  Indicates that group-by-expression is a prefix column of a subtotal row.

**See also**

- “The ROLLUP clause” [SQL Anywhere Server - SQL Usage]
- “The CUBE clause” [SQL Anywhere Server - SQL Usage]
- “GROUP BY GROUPING SETS” [SQL Anywhere Server - SQL Usage]
- “SELECT statement” on page 955
- “Detection of NULLs using the GROUPING function” [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**

- **SQL/2008**  The GROUPING function is part of optional SQL/2008 language feature T431, "Extended grouping capabilities".

**Example**

For examples of this function, see “Detection of NULLs using the GROUPING function” [SQL Anywhere Server - SQL Usage].

HASH function [String]

Returns the specified value in hashed form.

**Syntax**

HASH( string-expression[, algorithm ] )

**Parameters**

- **string-expression**  The value to be hashed. This parameter is case sensitive, even in case-insensitive databases.
The algorithm to use for the hash. Possible values include: CRC32, MD5, SHA1, SHA1_FIPS, SHA256, SHA256_FIPS. By default, the MD5 algorithm is used. FIPS-certified algorithms require a separate license.

Returns
Following are the return types, depending on the algorithm used:

- CRC32 returns a hexadecimal string. Use the HEXToint function to convert the hexadecimal string to a 32-bit integer.
- MD5 returns a VARCHAR(32)
- SHA1 returns a VARCHAR(40)
- SHA1_FIPS returns a VARCHAR(40)
- SHA256 returns a VARCHAR(64)
- SHA256_FIPS returns a VARCHAR(64)

Remarks
Using a hash converts the value to a byte sequence that is unique to each value passed to the function.

If the database server was started with the -fips option, the algorithm used, or the behavior, may be different, as follows:

- SHA1_FIPS is used if SHA1 is specified
- SHA256_FIPS is used if SHA256 is specified.
- an error is returned if MD5 is specified.
- the CRC32 algorithm, which is not FIPS-certified, is allowed in FIPS mode because it is not considered a cryptographic algorithm.

Caution
All the algorithms are one-way hashes. It is not possible to re-create the original string from the hash.

See also
- “String functions” on page 152
- “-fips database server option” [SQL Anywhere Server - Database Administration]
- “SQL Anywhere security option” [SQL Anywhere 16 - Introduction]
- “HEXToint function [Data type conversion]” on page 258

Standards and compatibility
- SQL/2008 Vendor extension.
Example

The following example creates a table called user_info to store information about the users of an application, including their user ID and password. One row is also inserted into the table. The password is hashed using the HASH function and the SHA256 algorithm. Storing hashed passwords in this way can be useful if you do not want to store passwords in clear text, yet you have an external application that needs to compare passwords.

```
CREATE TABLE user_info (  
  employee_id   INTEGER NOT NULL PRIMARY KEY,
  user_name CHAR(80),
  user_pwd CHAR(80) );

INSERT INTO user_info
VALUES ( '1', 's_phillips', HASH( 'mypass', 'SHA256' ) );
```

HEXTOBIN function [Data type conversion]

Returns the LONG BINARY equivalent of a hexadecimal string.

Syntax

```
HEXTOBIN( hexadecimal-string )
```

Parameters

- `hexadecimal-string`: The string to be converted to a binary string.

Returns

The HEXTOBIN function returns a LONG BINARY string. If the number of characters in the input is odd, it is left-padded with a zero. The length of the result is the length of the input string divided by 2. If the input string contains any non-hexadecimal characters, an error is returned.

Remarks

The HEXTOBIN function accepts string literals or variables consisting only of digits and the uppercase or lowercase letters A-F.

The BINTOHEX, CAST, CONVERT, HEXTOBIN, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values. For more information about using these functions, see “Converting to and from hexadecimal values” on page 7.

See also

- “BINTOHEX function [Data type conversion]” on page 166

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns a binary string containing 0x313233:

```
SELECT HEXTOBIN( '313233' );
```
HEXTOINT function [Data type conversion]

Returns the decimal integer equivalent of a hexadecimal string.

The CAST, CONVERT, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values. For more information about using these functions, see “Converting to and from hexadecimal values” on page 7.

Syntax

HEXTOINT( hexadecimal-string )

Parameters

- **hexadecimal-string**  The string to be converted to an integer.

Returns

The HEXTOINT function returns as INT the platform-independent SQL INTEGER equivalent of the hexadecimal string. The hexadecimal value represents a negative integer if the 8th digit from the right is one of the digits 8-9 and the uppercase or lowercase letters A-F and the previous leading digits are all uppercase or lowercase letter F. The following is not a valid use of HEXTOINT since the argument represents a positive integer value that cannot be represented as a signed 32-bit integer:

```
SELECT HEXTOINT( '0x0080000001' );
```

Remarks

The HEXTOINT function accepts string literals or variables consisting only of digits and the uppercase or lowercase letters A-F, with or without a 0x prefix. The following are all valid uses of HEXTOINT:

```
SELECT HEXTOINT( '0xFFFFFFFF' );
SELECT HEXTOINT( '0x00000100' );
SELECT HEXTOINT( '100' );
SELECT HEXTOINT( '0xffffffff80000001' );
```

The HEXTOINT function removes the 0x prefix, if present. If the data exceeds 8 digits, it must represent a value that can be represented as a signed 32-bit integer value.

This function supports NCHAR inputs and/or outputs.

See also

- “INTTOHEX function [Data type conversion]” on page 274

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statement returns the value 420:

```
SELECT HEXTOINT( '1A4' );
```
HOUR function [Date and time]

Returns the hour component of a TIMESTAMP value.

Syntax
HOUR( timestamp-expression )

Parameters
● timestamp-expression A TIMESTAMP value.

Returns
SMALLINT

Remarks
The value returned is the hour portion of the TIMESTAMP expression, a SMALLINT value between 0 and 23.

Standards and compatibility
● SQL/2008 Vendor extension.

Example
The following statement returns the value 21:

```
SELECT HOUR( '1998-07-09 21:12:13' );
```
The number of hours to be added to time-or-timestamp-expression. If integer-expression is negative, the appropriate number of hours is subtracted from time-or-timestamp-expression.

For information about casting data types, see “CAST function [Data type conversion]” on page 174.

**Returns**

INTEGER with Syntax 1 or Syntax 2.

TIME or TIMESTAMP with Syntax 3.

**Remarks**

The result of the HOURS function depends on its arguments.

- **Syntax 1** If you pass a single timestamp-expression to the HOURS function, it will return the number of hours between midnight 0000-02-29 and timestamp-expression as an INTEGER.

  **Note** 0000-02-29 is not meant to imply an actual date; it is the default TIMESTAMP value used by the HOURS function.

- **Syntax 2** If you pass two TIMESTAMP values to the HOURS function, the function returns the integer number of hours between them.

- **Syntax 3** If you pass a TIMESTAMP value and an INTEGER value to the HOURS function, the function returns the TIMESTAMP result of adding the integer number of hours to time-or-timestamp-expression argument. Similarly, if you pass a TIME value as the first argument, a TIME value is returned as the result. Syntax 3 does not support implicit conversion of the first argument. It may be necessary to explicitly cast the first argument to a DATE, TIME or TIMESTAMP value. If the first argument is a DATE, midnight is assumed for the time portion.

  Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.

**See also**

- “DATEDIFF function [Date and time]” on page 205
- “DATEADD function [Date and time]” on page 204

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statements return the value 4, signifying that the second TIMESTAMP value is four hours after the first. It is recommended that you use the second example (DATEDIFF).

```
SELECT HOURS( '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
SELECT DATEDIFF( hour, '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
```

The following statement returns the value 17517342:
SELECT HOURS( '1998-07-13 06:07:12' );

The following statements return the datetime 1999-05-13 02:05:07.000. It is recommended that you use the second example (DATEADD).

SELECT HOURS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 );

SELECT DATEADD( hour, 5, '1999-05-12 21:05:07' );

**HTML_DECODE function [Miscellaneous]**

Decodes special character entities that appear in HTML literal strings.

**Syntax**

```sql
HTML_DECODE( string )
```

**Parameters**

- `string` Arbitrary literal string used in an HTML document.

**Returns**

LONG VARCHAR or LONG NVARCHAR.

**Remarks**

This function returns the string argument after making the appropriate substitutions. The following table contains a sampling of the acceptable character entities.

<table>
<thead>
<tr>
<th>Characters</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&amp;xhexadecimal-number;</td>
<td>Unicode codepoint, specified as a hexadecimal number. For example, ' returns a single apostrophe.</td>
</tr>
<tr>
<td>&amp;decimal-number;</td>
<td>Unicode codepoint, specified as a decimal number. For example, ™ returns the trademark symbol.</td>
</tr>
</tbody>
</table>

When a Unicode codepoint is specified, if the value can be converted to a character in the database character set, it is converted to a character. Otherwise, it is returned uninterpreted.

See also

● “HTML_ENCODE function [Miscellaneous]” on page 262
● “Web services functions” on page 151
● “Web services system procedures” on page 1085

Standards and compatibility

● SQL/2008 Vendor extension.

Examples

The following statement returns the string <p>The piano was made for 'Steinway & Sons'.</p>:

```sql
SELECT HTML_DECODE('<p>The piano was made ' || 'by &lsquo;Steinway &amp; Sons&rsquo;.' || '</p>')
```

The following statement returns the string <p>It cost €85.000,000.</p>:

```sql
SELECT HTML_DECODE('&lt;p&gt;It cost &euro;85.000,000.&lt;/p&gt;')
```

**HTML_ENCODE function [Miscellaneous]**

Encodes special characters within strings to be inserted into HTML documents.

Syntax

```
HTML_ENCODE( string )
```

Parameters

● **string** Arbitrary string to be used in an HTML document.

Returns

LONG VARCHAR or LONG NVARCHAR.

Remarks

This function returns the string argument after making the following set of substitutions:

<table>
<thead>
<tr>
<th>Characters</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
</tbody>
</table>
This function supports NCHAR inputs and/or outputs.

See also
- “HTML_DECODE function [Miscellaneous]” on page 261
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Standards and compatibility
- SQL/2008    Vendor extension.

Examples
The following example returns the string '\lt;!DOCTYPE HTML PUBLIC &quot;~//W3C//DTD HTML 4.01//EN&quot;&gt;'.

    SELECT HTML_ENCODE('<!DOCTYPE HTML PUBLIC "~//W3C//DTD HTML 4.01//EN">')

HTTP_BODY function [Web service]
Returns the body of the HTTP request in binary form. For example, in a POST request, this is the raw POST data.

Syntax
HTTP_BODY()

Parameters
None

Returns
LONG VARCHAR containing the body of the HTTP request in binary form; no character set conversion is performed on it.

Remarks
If the request body does not exist, or if the function is not called from a web service, a NULL value is returned.

This function is useful within the PHP external environment.
HTTP_DECODE function [Web service]

Decodes HTTP encoded strings. This is also known as URL decoding.

Syntax

HTTP_DECODE( string )

Parameters

- string  Arbitrary string taken from a URL or URL encoded request body.

Returns

LONG VARCHAR or LONG NVARCHAR

Remarks

This function returns the string argument after replacing all character sequences of the form %nn, where nn is a hexadecimal value, with the character with code nn. In addition, all plus signs (+) are replaced with spaces.

See also

- “HTTP_ENCODE function [Web service]” on page 264
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Standards and compatibility

- SQL/2008  Vendor extension.

Examples

The following statement returns the string http://dcx.sybase.com:

```
SELECT HTTP_DECODE('http%3A%2F%2Fdcx.sybase.com')
```

HTTP_ENCODE function [Web service]

Encodes strings for use with HTTP. This is also known as URL encoding.
Syntax

HTTP_ENCODE( string )

Parameters

- **string**  Arbitrary string to be encoded for HTTP transport.

Returns

LONG VARCHAR or LONG NVARCHAR

Remarks

This function returns the string argument after making the following set of substitutions. In addition, all characters with hexadecimal codes less than 20 or greater than 7E are replaced with \%nn, where nn is the character code.

<table>
<thead>
<tr>
<th>Character</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>%20</td>
</tr>
<tr>
<td>&quot;</td>
<td>%22</td>
</tr>
<tr>
<td>#</td>
<td>%23</td>
</tr>
<tr>
<td>%</td>
<td>%25</td>
</tr>
<tr>
<td>&amp;</td>
<td>%26</td>
</tr>
<tr>
<td>,</td>
<td>%2C</td>
</tr>
<tr>
<td>;</td>
<td>%3B</td>
</tr>
<tr>
<td>&lt;</td>
<td>%3C</td>
</tr>
<tr>
<td>&gt;</td>
<td>%3E</td>
</tr>
<tr>
<td>[</td>
<td>%5B</td>
</tr>
<tr>
<td>\</td>
<td>%5C</td>
</tr>
<tr>
<td>]</td>
<td>%5D</td>
</tr>
<tr>
<td>`</td>
<td>%60</td>
</tr>
<tr>
<td>{</td>
<td>%7B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>%7D</td>
</tr>
</tbody>
</table>
### CHARACTER AND SUBSTITUTION

<table>
<thead>
<tr>
<th>Character codes</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>nn that are less than 0x20 and greater than 0x7f</td>
<td>%nn</td>
</tr>
</tbody>
</table>

This function supports NCHAR inputs and/or outputs.

### See also

- “HTTP_DECODE function [Web service]” on page 264
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

### Standards and compatibility

- **SQL/2008**  Vendor extension.

### Examples

The following statement returns the string `/opt%26id=123%26text='oid:c%09d%20ef'`:

```sql
SELECT HTTP_ENCODE('/opt&id=123&text='oid:c\x09d ef''')
```

---

## HTTP_HEADER function [Web service]

Returns the value of an HTTP request header.

### Syntax

```
HTTP_HEADER( header-field-name )
```

### Parameters

- **header-field-name**  The name of an HTTP request header field.

### Returns

LONG VARCHAR.

### Note

The result data type is a LONG VARCHAR. If you use HTTP_HEADER in a SELECT INTO statement, you must have an Unstructured Data Analytics Option license or use CAST and set HTTP_HEADER to the correct data type and size.

### Remarks

This function returns the value of the named HTTP request header field, or NULL if it does not exist or if it is not called from an HTTP service. It is used when processing an HTTP request via a web service.

Some headers that may be of interest when processing an HTTP web service request include the following:

- **Cookie**  The cookie value(s), if any, stored by the client, that are associated with the requested URI.
- **Referer**  The URL of the page (for example, http://documents.sample.com:80/index.html) that contained the link to the requested URI.

- **Host**  The Internet host name or IP address and port number of the resource being requested, as obtained from the original URI given by the user or referring resource (for example, webserver.sample.com:8082).

- **User-Agent**  The name of the client application (for example, Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0).

- **Accept-Encoding**  A list of encodings for the response that are acceptable to the client application (for example, gzip, deflate).

  More information about these headers is available at http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html.

The following special headers allow access to the elements within the request line of a client request.

- **@HttpMethod**  Returns the type of request being processed. Possible values include DELETE, HEAD, GET, PUT, or POST.

- **@HttpURI**  The full URI of the request, as it was specified in the HTTP request (for example, /myservice?id=-123&version=109&lang=en).

- **@HttpVersion**  The HTTP version of the request (for example, HTTP/1.0, or HTTP/1.1).

- **@HttpQueryString**  Returns the query portion of the requested URI if it exists (for example, id=-123&version=109&lang=en).

See also
- “NEXT_HTTP_HEADER function [Web service]” on page 306
- “sa_set_http_header system procedure” on page 1239
- “sa_http_header_info system procedure” on page 1154
- “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Standards and compatibility
- **SQL/2008**  Vendor extension.

Example

The following statement retrieves the Cookie header value when used within a stored procedure that is called by an HTTP web service:

```sql
SET cookie_value = HTTP_HEADER( 'Cookie' );
```

The following statement displays the name and values of the HTTP request headers in the database server messages window when used within a stored procedure that is called by an HTTP web service:
BEGIN
    declare header_name long varchar;
    declare header_value long varchar;
    set header_name  = NULL;
    header_loop:
    LOOP
        SET header_name = NEXT_HTTP_HEADER( header_name );
        IF header_name IS NULL THEN
            LEAVE header_loop
        END IF;
        SET header_value = HTTP_HEADER( header_name );
        MESSAGE 'HEADER: ', header_name, '=",
        header_value TO CONSOLE;
    END LOOP;
END;

HTTP_RESPONSE_HEADER function [Web service]

Returns the value of an HTTP response header.

Syntax

    HTTP_RESPONSE_HEADER( header-field-name )

Parameters

- **header-field-name**  The name of an HTTP response header field.

Returns

    LONG VARCHAR

Remarks

This function returns the value of the named HTTP response header field, or NULL if a header for the
given *header-field-name* does not exist or if it is not called from an HTTP service.

Some headers that may be of interest when processing an HTTP web service response include the
following:

- **Connection**  The Connection field allows the sender to specify options that are desired for that
  particular connection. In a SQL Anywhere HTTP server response, the option is always "close".

- **Content-Length**  The Content-Length field indicates the size of the response body, in decimal
  number of octets.

- **Content-Type**  The Content-Type field indicates the media type of the body sent to the recipient.
  For example: *text/xml*

- **Date**  The Date field represents the date and time at which the response was originated.

- **Expires**  The Expires field gives the date and time after which the response is considered stale.

- **Location**  The Location field is used to redirect the recipient to a location for completion of the
  request or identification of a new resource.
● **Server**  The Server field contains information about the software used by the origin server to handle the request. In a SQL Anywhere HTTP server response, the web server name together with the version number is returned.

● **Transfer-Encoding**  The Transfer-Encoding field indicates what (if any) type of transformation has been applied to the message body to safely transfer it between the sender and the recipient.

● **User-Agent**  The User-Agent field contains information about the user agent originating the request. In a SQL Anywhere HTTP server response, the web server name together with the version number is returned.

● **WWW-Authenticate**  The WWW-Authenticate field is included in 401 (Unauthorized) response messages.

More information about these headers is available at http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html.

The following special header allows access to the status within the response of a server response.

● **@HttpStatus**  Returns the status code of the processed request.

**See also**

- “NEXT_HTTP_RESPONSE_HEADER function [Web service]” on page 308
- “sa_set_http_header system procedure” on page 1239
- “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

**Example**

The following statement displays the name and values of the HTTP response headers in the database server messages window when used within a stored procedure that is called by an HTTP web service:

```
BEGIN
    declare header_name long varchar;
    declare header_value long varchar;
    set header_name = NULL;
    header_loop:
    LOOP
        SET header_name = NEXT_HTTP_RESPONSE_HEADER( header_name );
        IF header_name IS NULL THEN
            LEAVE header_loop
        END IF;
        SET header_value = HTTP_RESPONSE_HEADER( header_name );
        MESSAGE 'RESPONSE HEADER: ', header_name, '=', header_value TO CONSOLE;
    END LOOP;
```

HTTP_VARIABLE function [Web service]

Returns the value of an HTTP variable.

Syntax

HTTP_VARIABLE( var-name [ , instance [ , attribute ] ] )

Parameters

- **var-name** The name of an HTTP variable.

- **instance** If more than one variable has the same name, the instance number of the field instance, or NULL to get the first one. Useful for SELECT lists that permit multiple selections.

- **attribute** In a multi-part request, the attribute can specify a header field name which returns the value of the header for the multi-part name.

When an attribute is not specified, the returned value is %-decoded and character-set translated to the database character set. UTF %-encoded data is supported in this mode.

The attribute can also be one of the following modes:

- `'@BINARY'` Returns a x-www-form-urlencoded binary data value. This mode indicates that the returned value is %-decoded and not character-set translated. UTF-8 %-encoding is not supported in this mode since %-encoded data are simply decoded into their equivalent byte representation.

- `'@TRANSPORT'` Returns the raw HTTP transport form of the value, where %-encodings are preserved.

Returns

LONG VARCHAR.

Remarks

This function returns the value of the named HTTP variable. It is used when processing an HTTP request within a web service.

If var-name does not exist, the return value is NULL.

When the web service request is a POST, and the variable data is posted as multipart/form-data, the HTTP server receives HTTP headers for each individual variable. When the attribute parameter is specified, the HTTP_VARIABLE function returns the associated multipart/form-data header value from the POST request for the particular variable. For a variable representing a file, an attribute of Content-Disposition, Content-Type, and @BINARY would return the filename, media-type, and file contents respectively.

Normally, all input data goes through character set translation between the client (for example, a browser) character set, and the character set of the database. However, if @BINARY is specified for attribute, the variable value is returned without going through character set translation or %-decoding. This may be useful when receiving binary data, such as image data, from a client.
This function returns NULL when the specified instance does not exist or when the function is called from outside of an execution of a web service.

See also

- “NEXT_HTTP_VARIABLE function [Web service]” on page 309
- “sa_http_variable_info system procedure” on page 1158
- “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Standards and compatibility

- SQL/2008   Vendor extension.

Examples

The following statement retrieves the values of the HTTP variables indicated in the sample URL when used within a stored procedure that is called by an HTTP web service:

```sql
-- http://sample.com/demo/ShowDetail?product_id=300&customer_id=101
BEGIN
  DECLARE v_customer_id LONG VARCHAR;
  DECLARE v_product_id LONG VARCHAR;
  SET v_customer_id = HTTP_VARIABLE( 'customer_id' );
  SET v_product_id = HTTP_VARIABLE( 'product_id' );
  CALL ShowSalesOrderDetail( v_customer_id, v_product_id );
END;
```

The following statements request the Content-Disposition and Content-Type headers of the image variable when used within a stored procedure that is called by an HTTP web service:

```sql
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
```

The following statement requests the value of the image variable in its current character set without going through character set translation when used within a stored procedure that is called by an HTTP web service:

```sql
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );
```

**IDENTITY function [Miscellaneous]**

Generates integer values, starting at 1, for each successive row in a query. Its implementation is identical to that of the NUMBER function.

**Syntax**

```
IDENTITY( expression )
```

**Parameters**

- `expression` An expression. The expression is parsed, but is ignored during the execution of the function.
Returns

INT

Remarks

The description of the IDENTITY function is the same as the description of the NUMBER function.

See also

- “NUMBER function [Miscellaneous]” on page 313

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns a sequentially-numbered list of employees:

```
SELECT IDENTITY( 10 ), Surname FROM GROUPO.Employees;
```

**IFNULL function [Miscellaneous]**

If the first expression is the NULL value, then the value of the second expression is returned. If the first expression is not NULL, the value of the third expression is returned. If the first expression is not NULL and there is no third expression, NULL is returned.

Syntax

```
IFNULL( expression-1, expression-2 [ , expression-3 ] )
```

Parameters

- **expression-1** The expression to be evaluated. Its value determines whether expression-2 or expression-3 is returned.
- **expression-2** The return value if expression-1 is NULL.
- **expression-3** The return value if expression-1 is not NULL.

Returns

The data type returned depends on the data type of expression-2 and expression-3.

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns the value -66:

```
SELECT IFNULL( NULL, -66 );
```

The following statement returns NULL, because the first expression is not NULL and there is no third expression:
INDEX_ESTIMATE function [Miscellaneous]

Returns selectivity estimates from the index as a percentage calculated by the query optimizer, based on specified parameters.

Syntax

INDEX_ESTIMATE( column-name [ , value [ , relation-string ] ] )

Parameters

- **column-name** The column used in the estimate.
- **value** The value to which the column is compared. The default is NULL.
- **relation-string** The comparison operator used for the comparison, enclosed in single quotes. Possible values for this parameter are: '=', '>', '<', '>=', '<=', '!=', '<>', and '!='. The default is '='.

Returns

REAL

Remarks

This function returns selectivity estimates from the index for the predicate column-name relation-string value. If value is NULL and the relation string is '=', the selectivity is for the predicate column-name IS NULL. If value is NULL and the relation string is '!=' or '<>', the selectivity is for the predicate column-name IS NOT NULL.

See also

- “ESTIMATE function [Miscellaneous]” on page 236
- “ESTIMATE_SOURCE function [Miscellaneous]” on page 237
- “EXPERIENCE_ESTIMATE function [Miscellaneous]” on page 243

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns the percentage of EmployeeID values estimated to be greater than 200:

```
SELECT INDEX_ESTIMATE( EmployeeID, 200, '>' )
FROM GROUPO.Employees;
```

INSERTSTR function [String]

Inserts a string into another string at a specified position.
Syntax

\texttt{INSERTSTR( integer-expression, string-expression-1, string-expression-2 )}

Parameters

- \texttt{integer-expression}  The position after which the string is to be inserted. Use zero to insert a string at the beginning.
- \texttt{string-expression-1}  The string into which the other string is to be inserted.
- \texttt{string-expression-2}  The string to be inserted.

Returns

\texttt{LONG VARCHAR}

Remarks

This function supports NCHAR inputs and/or outputs.

See also

- “STUFF function [String]” on page 378
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the value \texttt{backoffice}:

\begin{verbatim}
SELECT INSERTSTR( 0, 'office ', 'back' );
\end{verbatim}

\textbf{INTTOHEX function [Data type conversion]}

Returns a string containing the hexadecimal equivalent of an integer.

Syntax

\texttt{INTTOHEX( integer-expression )}

Parameters

- \texttt{integer-expression}  The integer to be converted to hexadecimal.

Returns

\texttt{VARCHAR}

Remarks

The CAST, CONVERT, HEXTOINT, and INTTOHEX functions can be used to convert to and from hexadecimal values.
See also

- “HEXTOINT function [Data type conversion]” on page 258
- “Converting to and from hexadecimal values” on page 7

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the value 0000009c:

```
SELECT INTTOHEX( 156 );
```

**ISDATE function [Data type conversion]**

Tests if a string argument can be converted to a date.

**Syntax**

```
ISDATE( string )
```

**Parameters**

- `string`  The string to be analyzed to determine if the string represents a valid date.

**Returns**

INT

**Remarks**

If a conversion is possible, the function returns 1; otherwise, 0 is returned. If the argument is NULL, 0 is returned.

This function supports NCHAR inputs and/or outputs.
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example imports data from an external file, exports rows which contain invalid values, and copies the remaining rows to a permanent table:

```
CREATE GLOBAL TEMPORARY TABLE MyData(
    person VARCHAR(100),
    birth_date VARCHAR(30),
    height_in_cms VARCHAR(10)
) ON COMMIT PRESERVE ROWS;
LOAD TABLE MyData FROM 'exported.dat';
UNLOAD
    SELECT * FROM MyData
    WHERE ISDATE( birth_date ) = 0
    OR ISNUMERIC( height_in_cms ) = 0
    TO 'badrows.dat';
INSERT INTO PermData
    SELECT person, birth_date, height_in_cms
    FROM MyData
    WHERE ISDATE( birth_date ) = 1
    AND ISNUMERIC( height_in_cms ) = 1;
COMMIT;
DROP TABLE MyData;
```

**ISENCRYPTED function [System]**

Determines if a string is encrypted using the ENCRYPT function and the specified key.

**Syntax**

```
ISENCRYPTED( string, key[, algorithm ])
```
Returns

INT

Parameters

- **string** The string to be analyzed to determine if it is encrypted. This parameter is case sensitive, even in case-insensitive databases.

- **key** The encryption key used to encrypt the string. This parameter is case sensitive, even in case-insensitive databases.

- **algorithm** This optional parameter specifies the algorithm used when the string was encrypted. Supported algorithms include: AES, AES256, AES_FIPS, and AES256_FIPS.

  You can specify one of the FIPS-certified algorithms for algorithm on any platform that supports FIPS-certified encryption.

  **Note**
  RSA and FIPS-certified encryption are not available on all platforms. For information about which platforms support which encryption method, see [http://www.sybase.com/detail?id=1061806](http://www.sybase.com/detail?id=1061806).

  If algorithm is not specified, AES is used by default. If the database server was started using the -fips server option, the default is AES_FIPS.

Remarks

ISENCRYPTED returns 1 when the input string is encrypted with the specified key; otherwise it returns 0.

See also

- “ENCRYPT function [String]” on page 226
- “DECRYPT function [String]” on page 220

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns the value 1 because the input string is encrypted:

```
SELECT ISENCRYPTED( ENCRYPT ('test_string', 'key'), 'key');
```

**ISNULL function [Miscellaneous]**

Returns the first non-NULL expression from a list. This function is identical to the COALESCE function.

**Syntax**

```
ISNULL( expression, expression [, ... ] )
```
### Parameters

- **expression**  
  An expression to be tested against NULL.

  At least two expressions must be passed into the function, and all expressions must be comparable.

### Returns

The return type for this function depends on the expressions specified. That is, when the database server evaluates the function, it first searches for a data type in which all the expressions can be compared. When found, the database server compares the expressions and then returns the result in the type used for the comparison. If the database server cannot find a common comparison type, an error is returned.

### See also

- “COALESCE function [Miscellaneous]” on page 179

### Standards and compatibility

- **SQL/2008**  
  Vendor extension.

### Example

The following statement returns the value -66:

```sql
SELECT ISNULL( NULL , -66, 55, 45, NULL, 16 );
```

### ISNUMERIC function [Miscellaneous]

 Determines if a string argument is a valid number.

#### Syntax

```sql
ISNUMERIC( string )
```

#### Parameters

- **string**  
  The string to be analyzed to determine if the string represents a valid number.

#### Returns

INT

#### Remarks

ISNUMERIC returns 1 when the input string evaluates to a valid integer or floating-point number; otherwise it returns 0. The function also returns 0 if the string contains only blanks or is NULL.

Following are values that also cause the ISNUMERIC function to return 0:

- Values that use the letter d or D as the exponent separator. For example, 1d2.
- Special values such as NAN, 0x12, INF, and INFINITY.
- NULL (for example, `SELECT ISNUMERIC( NULL );`).
Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following example imports data from an external file, exports rows that contain invalid values, and copies the remaining rows to a permanent table. In this example, the ISNUMERIC statement validates that the values in `height_in_cms` values are numeric.

```sql
CREATE GLOBAL TEMPORARY TABLE MyData(
  person VARCHAR(100),
  birth_date VARCHAR(30),
  height_in_cms VARCHAR(10)
) ON COMMIT PRESERVE ROWS;
LOAD TABLE MyData FROM 'exported.dat';
UNLOAD
  SELECT *
  FROM MyData
  WHERE ISDATE(birth_date) = 0
  OR ISNUMERIC(height_in_cms) = 0
TO 'badrows.dat';
INSERT INTO PermData
  SELECT person, birth_date, height_in_cms
  FROM MyData
  WHERE ISDATE(birth_date) = 1
  AND ISNUMERIC(height_in_cms) = 1;
COMMIT;
DROP TABLE MyData;
```

**LAST_VALUE function [Aggregate]**

Returns values from the last row of a window.

**Syntax**

```
LAST_VALUE( [ ALL ] expression[ { RESPECT | IGNORE } NULLS ] )
OVER ( window-spec )
```

*window-spec*: see the Remarks section below

**Parameters**

- **expression** The expression to evaluate. For example, a column name.

**Returns**

Data type of the argument.

**Remarks**

The LAST_VALUE function allows you to select the last value (according to some ordering) in a table, without having to use a self-join. This is valuable when you want to use the last value as the baseline in calculations.

The LAST_VALUE function takes the last record from the partition after doing the ORDER BY. Then, the `expression` is computed against the last record and results are returned.
If IGNORE NULLS is specified, the last non-NULL value of expression is returned. If RESPECT NULLS is specified (the default), the last value is returned whether or not it is NULL.

The LAST_VALUE function is different from most other aggregate functions in that it can only be used with a window specification.

Elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause. See “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also
- “Window aggregate functions” [SQL Anywhere Server - SQL Usage]
- “FIRST_VALUE function [Aggregate]” on page 246

Standards and compatibility
- SQL/2008  Vendor extension.

SQL Anywhere supports SQL/2008 language feature F441, "Extended set function support", which permits operands of window functions to be arbitrary expressions that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the LAST_VALUE function, combined with an outer reference. For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example

The following example returns the salary of each employee, plus the name of the employee with the highest salary in the same department:

```sql
SELECT GivenName + ' ' + Surname AS employee_name,
       Salary, DepartmentID,
       LAST_VALUE( employee_name ) OVER Salary_Window AS highest_paid
FROM GROUPO.Employees
WINDOW Salary_Window AS ( PARTITION BY DepartmentID ORDER BY Salary
                          RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING );
```

<table>
<thead>
<tr>
<th>employee_name</th>
<th>Salary</th>
<th>DepartmentID</th>
<th>highest_paid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael Lynch</td>
<td>24903</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Joseph Barker</td>
<td>27290</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>employee_name</td>
<td>Salary</td>
<td>DepartmentID</td>
<td>highest_paid</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------</td>
<td>--------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Sheila Romero</td>
<td>27500</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Felicia Kuo</td>
<td>28200</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Jeannette Bertrand</td>
<td>29800</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Jane Braun</td>
<td>34300</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Anthony Rebeiro</td>
<td>34576</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Charles Crowley</td>
<td>41700</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Jose Martinez</td>
<td>55500.8</td>
<td>500</td>
<td>Jose Martinez</td>
</tr>
<tr>
<td>Doug Charlton</td>
<td>28300</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Elizabeth Lambert</td>
<td>29384</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Joyce Butterfield</td>
<td>34011</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Robert Nielsen</td>
<td>34889</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Alex Ahmed</td>
<td>34992</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>Ruth Wetherby</td>
<td>35745</td>
<td>400</td>
<td>Scott Evans</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Jose Martinez makes the highest salary in department 500, and Scott Evans makes the highest salary in department 400.

**LCASE function [String]**

Converts all characters in a string to lowercase.

**Syntax**

```
LCASE( string-expression )
```

**Parameters**

- `string-expression`  The string to be converted to lowercase.
Returns

- CHAR
- NCHAR
- LONG VARCHAR
- VARCHAR
- NVARCHAR

Remarks

The LCASE function is identical to the LOWER function.

See also

- “LOWER function [String]” on page 290
- “UCASE function [String]” on page 396
- “UPPER function [String]” on page 399
- “String functions” on page 152

Standards and compatibility

- SQL/2008 Vendor extension. The equivalent function LOWER is a core feature of the SQL/2008 standard.

Example

The following statement returns the value chocolate:

```sql
SELECT LCASE( 'ChoCOlatE' );
```

**LEFT function [String]**

Returns multiple characters from the beginning of a string.

Syntax

```sql
LEFT( string-expression, integer-expression )
```

Parameters

- **string-expression** The string.
- **integer-expression** The number of characters to return.

Returns

- LONG VARCHAR
- LONG NVARCHAR

Remarks

If the string contains multibyte characters, and the proper collation is being used, the number of bytes returned may be greater than the specified number of characters.
You can specify an integer-expression that is larger than the value in the argument string expression. In this case, the entire value is returned.

This function supports NCHAR inputs and/or outputs. Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics.

See also
- “RIGHT function [String]” on page 350
- “String functions” on page 152

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement returns the first 5 characters of each Surname value in the Customers table:

```
SELECT LEFT( Surname, 5) FROM GROUPO.Customers;
```
See also

- “BYTE_LENGTH function [String]” on page 171
- “International languages and character sets” [SQL Anywhere Server - Database Administration]
- “String functions” on page 152

Standards and compatibility

- SQL/2008   The LENGTH function is a vendor extension; however, its semantics are identical to those of the CHAR_LENGTH function in the SQL/2008 standard. Using LENGTH over a string expression of type NCHAR comprises part of optional SQL/2008 language feature F421.

Example

The following statement returns the value 9:

```
SELECT LENGTH( 'chocolate' );
```
LIST function [Aggregate]

Returns a delimited list of values for every row in a group.

Syntax

```
LIST(
    [ALL | DISTINCT] string-expression
    [, delimiter-string]
    [ORDER BY order-by-expression [ ASC | DESC ], ... ]
)
```

Parameters

- **string-expression**  A string expression, usually a column name. When ALL is specified (the default), for each row in the group, the value of `string-expression` is added to the result string, with values separated by `delimiter-string`. When DISTINCT is specified, only unique `string-expression` values are added.

- **delimiter-string**  A delimiter string for the list items. The default setting is a comma. There is no delimiter if a value of NULL or an empty string is supplied. The `delimiter-string` must be a constant.

- **order-by-expression**  Order the items returned by the function. There is no comma preceding this argument, which makes it easy to use in the case where no `delimiter-string` is supplied.

`order-by-expression` cannot be an integer literal. However, it can be a variable that contains an integer literal.

When an ORDER BY clause contains constants, they are interpreted by the optimizer and then replaced by an equivalent ORDER BY clause. For example, the optimizer interprets ORDER BY 'a' as ORDER BY expression.

A query block containing more than one aggregate function with valid ORDER BY clauses can be executed if the ORDER BY clauses can be logically combined into a single ORDER BY clause. For example, the following clauses:

```
ORDER BY expression1, 'a', expression2
ORDER BY expression1, 'b', expression2, 'c', expression3
```

are subsumed by the clause:

```
ORDER BY expression1, expression2, expression3
```

Returns

- LONG VARCHAR
- LONG NVARCHAR

Remarks

The LIST function returns the concatenation (with delimiters) of all the non-NULL values of X for each row in the group. If there does not exist at least one row in the group with a definite X-value, then LIST( X ) returns the empty string.

NULL values and empty strings are ignored by the LIST function.
A LIST function cannot be used as a window function, but it can be used as input to a window function.

This function supports NCHAR inputs and/or outputs.

Standards and compatibility

- **SQL/2008**  Vendor extension.

SQL Anywhere supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the LIST function, combined with an outer reference. See “Troubleshooting: Aggregate functions and outer references” [SQL Anywhere 16 - Changes and Upgrading].

For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

See also

- “sa_split_list system procedure” on page 1248
- “ARRAY_AGG function [Aggregate]” on page 159

Examples

The following statement returns the value 487 Kennedy Court, 547 School Street:

```
SELECT LIST( Street ) FROM GROUPO.Employees
WHERE GivenName = 'Thomas';
```

The following statement lists employee IDs. Each row in the result set contains a comma-delimited list of employee IDs for a single department.

```
SELECT LIST( EmployeeID )
FROM GROUPO.Employees
GROUP BY DepartmentID;
```

<table>
<thead>
<tr>
<th>LIST( EmployeeID )</th>
</tr>
</thead>
<tbody>
<tr>
<td>102,105,160,243,247,249,266,278,...</td>
</tr>
<tr>
<td>129,195,299,467,641,667,690,856,...</td>
</tr>
<tr>
<td>148,390,586,757,879,1293,1336,...</td>
</tr>
<tr>
<td>184,207,318,409,591,888,992,1062,...</td>
</tr>
<tr>
<td>191,703,750,868,921,1013,1570,...</td>
</tr>
</tbody>
</table>

The following statement sorts the employee IDs by the last name of the employee:
SELECT LIST( EmployeeID ORDER BY Surname ) AS "Sorted IDs"
FROM GROUPO.Employees
GROUP BY DepartmentID;

<table>
<thead>
<tr>
<th>Sorted IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1013,191,750,921,868,1658,...</td>
</tr>
<tr>
<td>1751,591,1062,1191,992,888,318,...</td>
</tr>
<tr>
<td>1336,879,586,390,757,148,1483,...</td>
</tr>
<tr>
<td>1039,129,1142,195,667,1162,902,...</td>
</tr>
<tr>
<td>160,105,1250,247,266,249,445,...</td>
</tr>
</tbody>
</table>

The following statement returns semicolon-separated lists. Note the position of the ORDER BY clause and the list separator:

SELECT LIST( EmployeeID, ';' ORDER BY Surname ) AS "Sorted IDs"
FROM GROUPO.Employees
GROUP BY DepartmentID;

<table>
<thead>
<tr>
<th>Sorted IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1013;191;750;921;868;1658;703;...</td>
</tr>
<tr>
<td>1751;591;1062;1191;992;888;318;...</td>
</tr>
<tr>
<td>1336;879;586;390;757;148;1483;...</td>
</tr>
<tr>
<td>1039;129;1142;195;667;1162;902;...</td>
</tr>
<tr>
<td>160;105;1250;247;266;249;445;...</td>
</tr>
</tbody>
</table>

Be sure to distinguish the previous statement from the following statement, which returns comma-separated lists of employee IDs sorted by a compound sort-key of ( Surname, ';' ):

SELECT LIST( EmployeeID ORDER BY Surname, ';' ) AS "Sorted IDs"
FROM GROUPO.Employees
GROUP BY DepartmentID;

LOCATE function [String]

Returns the position of one string within another.

Syntax

LOCATE( string-expression-1, string-expression-2 [, integer-expression ] )
Parameters

- **string-expression-1**  The string to be searched.

- **string-expression-2**  The string to be searched for. This string is limited to 255 bytes.

- **integer-expression**  The character position in the string to begin the search. The first character is position 1. If the starting offset is negative, the locate function returns the last matching string offset rather than the first. A negative offset indicates how much of the end of the string is to be excluded from the search. The number of bytes excluded is calculated as \((-1 \times \text{offset}) -1\).

Returns

INT

Remarks

If **integer-expression** is specified, the search starts at that offset into the string.

The first string can be a long string (longer than 255 bytes), but the second is limited to 255 bytes. If a long string is given as the second argument, the function returns a NULL value. If the string is not found, 0 is returned. Searching for a zero-length string will return 1. If any of the arguments are NULL, the result is NULL.

If multibyte characters are used, with the appropriate collation, then the starting position and the return value may be different from the byte positions.

This function supports NCHAR inputs and/or outputs.

See also

- “String functions” on page 152
- “CHARINDEX function [String]” on page 177

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the value 8:

```
SELECT LOCATE(
    'office party this week - rsvp as soon as possible',
    'party',
    2);
```

The following statement:

```
BEGIN
    DECLARE STR LONG VARCHAR;
    DECLARE POS INT;
    SET str = 'c:\test\functions\locate.sql';
    SET pos = LOCATE( str, '\', -1 );
    select str, pos,
       SUBSTR( str, 1, pos -1 ) AS path,
       SUBSTR( str, pos +1 ) AS filename;
END;
```
LOG function [Numeric]

Returns the natural logarithm of a number.

Syntax

```
LOG( numeric-expression )
```

Parameters

- `numeric-expression` The number.

Returns

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Remarks

The argument is an expression that returns the value of any built-in numeric data type.

See also

- “LOG10 function [Numeric]” on page 289

Standards and compatibility

- **SQL/2008** The SQL/2008 standard defines the natural logarithm function using the keyword LN. The natural logarithm function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the natural logarithm of 50:

```
SELECT LOG( 50 );
```

LOG10 function [Numeric]

Returns the base 10 logarithm of a number.

Syntax

```
LOG10( numeric-expression )
```
Parameters
● numeric-expression  The number.

Returns
This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the parameter is NULL, the result is NULL.

Remarks
The argument is an expression that returns the value of any built-in numeric data type.

See also
● “LOG function [Numeric]” on page 289

Standards and compatibility
● SQL/2008  Vendor extension.

Example
The following statement returns the base 10 logarithm for 50:

```
SELECT LOG10( 50 );
```

**LOWER function [String]**
Converts all characters in a string to lowercase. This function is identical to the LCASE function.

Syntax
```
LOWER( string-expression )
```

Parameters
● string-expression  The string to be converted to lowercase.

Returns
CHAR, VARCHAR, LONG VARCHAR, NCHAR, NVARCHAR, or LONG NVARCHAR corresponding to the data type of the argument.

Remarks
The LCASE function is identical to the LOWER function.

See also
● “LCASE function [String]” on page 281
● “UCASE function [String]” on page 396
● “UPPER function [String]” on page 399
● “String functions” on page 152
Standards and compatibility


Example

The following statement returns the value chocolate:

   SELECT LOWER( 'chOCOLate' );

LTRIM function [String]

Removes leading blanks from the string.

Syntax

   LTRIM( string-expression )

Parameters

- string-expression  The string to be trimmed.

Returns

- VARCHAR
- NVARCHAR
- LONG VARCHAR
- LONG NVARCHAR

Remarks

The actual length of the result is the length of the expression minus the number of characters removed. If all the characters are removed, the result is an empty string.

If the parameter can be null, the result can be null.

If the parameter is null, the result is the null value.

This function supports NCHAR inputs and/or outputs.

See also

- “RTRIM function [String]” on page 356
- “TRIM function [String]” on page 393
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

   The TRIM specifications defined by the SQL/2008 standard (LEADING and TRAILING) are supplied by the SQL Anywhere LTRIM and RTRIM functions respectively.
Example

The following statement returns the value Test Message with all leading blanks removed:

```
SELECT LTRIM( '     Test Message' );
```

**MAX function [Aggregate]**

Returns the maximum expression value found in each group of rows.

**Syntax 1**

```
MAX( [ ALL | DISTINCT ] expression )
```

**Syntax 2**

```
MAX( [ ALL ] expression ) OVER ( window-spec )
```

*window-spec*: see Syntax 2 instructions in the Remarks section below

**Parameters**

- [ ALL ] *expression*  
  The expression for which the maximum value is to be calculated. This is commonly a column name.

- DISTINCT *expression*  
  Returns the same as MAX( *expression* ), and is included for completeness.

**Returns**

The same data type as the argument.

**Remarks**

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of *window-spec* can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the *window-spec* definition of the WINDOW clause. See “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

This function supports NCHAR inputs and/or outputs.

For simple comparisons of two expressions, you can also use the GREATER function.
See also

- “GREATER function [Miscellaneous]” on page 254
- “MIN function [Aggregate]” on page 295
- “Troubleshooting: Aggregate functions and outer references” [SQL Anywhere 16 - Changes and Upgrading]

Standards and compatibility

- **SQL/2008** Core feature. When used as a window function (Syntax 2), MAX comprises part of optional SQL/2008 language feature T611, "Basic OLAP operations".

  The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional SQL language feature F561, "Full value expressions". SQL Anywhere also supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

  SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the MAX function, combined with an outer reference.

  For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example

The following statement returns the value 138948.000, representing the maximum salary in the Employees table:

```sql
SELECT MAX( Salary )
FROM GROUPO.Employees;
```

**MEDIAN function [Aggregate]**

Computes the median of a numeric expression for a set of rows.

Syntax 1

```sql
MEDIAN([ ALL | DISTINCT ] numeric-expression )
```

Syntax 2

```sql
MEDIAN([ ALL ] numeric-expression ) OVER ( window-spec )
```

*window-spec*: see Syntax 2 instructions in the Remarks section below

Parameters

- **numeric-expression** The expression whose median is calculated over a set of rows.
- **DISTINCT clause** Eliminates duplicate values before computing the median of the unique values in the input.
● **ALL clause**  Computes the median of all values (including duplicates) in the input. This is the default behavior.

**Returns**

The data type of the returned value is the same as that of the input value.

NULLs are ignored in the calculation of the median value. However, a NULL value is returned for a group that contains no rows.

**Remarks**

`numeric-expression` values can be of any numeric data type other than BIT. See “Numeric data types” on page 97.

The median of a finite list of numbers can be found by arranging all the observations from lowest value to highest value and picking the middle one. If there is an even number of observations, the median is not unique so MEDIAN returns the mean of the two middle values. At most, half the population have values less than the median, and half have values greater than the median. If both groups contain less than half the population, then some of the population is exactly equal to the median. For example, if $a < b < c$, then the median of the list $\{a, b, c\}$ is $b$. If $a < b < c < d$, then the median of the list $\{a, b, c, d\}$ is the mean of $b$ and $c$ ($\frac{b + c}{2}$).

If the result of the mean of the two middle elements has digits after the decimal place, they are truncated if the input data type cannot represent them. To avoid this truncation, cast the input to a numeric type that allows digits after the decimal place.

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the `window-spec` definition provided in “WINDOW clause” on page 1051.

`window-spec` can only be over a partition (it cannot contain a ROW or RANGE specification). DISTINCT is not supported if a WINDOW clause is used. CUBE, ROLLUP, and GROUPING SETS are supported with syntax 1.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

**See also**

- “SUM function [Aggregate]” on page 381
- “COUNT function [Aggregate]” on page 192

**Standards and compatibility**

- **SQL/2008**  Vendor extension. Window functions comprise optional SQL/2008 language feature T611, "Basic OLAP operations".

  SQL Anywhere supports SQL/2008 language feature F441, "Extended set function support", which permits operands of window functions to be arbitrary expressions that are not column references.
SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the MEDIAN function, combined with an outer reference. For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example
The following statement returns the median salary from the Employees table:

```
SELECT MEDIAN( Salary ) FROM GROUPO.Employees;
```

The following statement returns the median salary by state from the Employees table:

```
SELECT EmployeeID, Surname, Salary, State,
       MEDIAN( Salary ) OVER Salary_Window
FROM GROUPO.Employees
WINDOW Salary_Window AS ( PARTITION BY State )
ORDER BY State, Surname;
```

MIN function [Aggregate]
Returns the minimum expression value found in each group of rows.

Syntax 1
```
MIN( [ ALL | DISTINCT ] expression )
```

Syntax 2
```
MIN( [ ALL ] expression ) OVER ( window-spec )
```

`window-spec`: see Syntax 2 instructions in the Remarks section below

Parameters
- [ ALL ] expression The expression for which the minimum value is to be calculated. This is commonly a column name.
- DISTINCT expression Returns the same as MIN( expression ), and is included for completeness.

Returns
The same data type as the argument.

Remarks
Rows where expression is NULL are ignored. Returns NULL for a group containing no rows.

This function supports NCHAR inputs and/or outputs.

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. More information is available in the window-spec definition for the WINDOW clause.
For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

For simple comparisons of two expressions, you can also use the LESSER function. See “LESSER function [Miscellaneous]” on page 284.

See also

- “MAX function [Aggregate]” on page 292
- “WINDOW clause” on page 1051

Standards and compatibility

- SQL/2008 Core feature. When used as a window function (Syntax 2), MIN comprises part of optional SQL/2008 language feature T611, "Basic OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional SQL language feature F561, "Full value expressions". SQL Anywhere also supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the MIN function, combined with an outer reference. See “Troubleshooting: Aggregate functions and outer references” [SQL Anywhere 16 - Changes and Upgrading].

For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example

The following statement returns the value 24903.000, representing the minimum salary in the Employees table:

```
SELECT MIN( Salary )
FROM GROUPO.Employees;
```

MINUTE function [Date and time]

Returns the minute component of a TIMESTAMP value.

Syntax

```
MINUTE( timestamp-expression )
```

Parameters

- `timestamp-expression` The TIMESTAMP value.
Returns
SMALLINT

Remarks
The value returned is the minute portion of the TIMESTAMP expression, a SMALLINT value between 0 and 59.

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement returns the value 22:

`SELECT MINUTE( '1998-07-13 12:22:34' );`

**MINUTES function [Date and time]**
Manipulates a TIMESTAMP or returns the number of minutes between two TIMESTAMP values. See the Remarks section below.

Syntax 1
`MINUTES( timestamp-expression )`

Syntax 2
`MINUTES( timestamp-expression, timestamp-expression )`

Syntax 3
`MINUTES( timestamp-or-time-expression, integer-expression )`

Parameters
- `timestamp-expression`  An expression of type TIMESTAMP.
- `timestamp-or-time-expression`  An expression of type TIME or TIMESTAMP.
- `integer-expression`  The number of minutes to be added to `timestamp-or-time-expression`. If `integer-expression` is negative, the appropriate number of minutes is subtracted from `timestamp-or-time-expression`.

Returns
INTEGER with Syntax 1 or Syntax 2.

TIME or TIMESTAMP with Syntax 3.

Remarks
The result of the MINUTES function depends on its arguments.
• **Syntax 1**  If you pass a single *timestamp-expression* to the MINUTES function, it will return the number of minutes between midnight 0000-02-29 and *timestamp-expression* as an INTEGER.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-02-29 is not meant to imply an actual date; it is the default date used by the MINUTES function.</td>
</tr>
</tbody>
</table>

• **Syntax 2**  If you pass two TIMESTAMP values to the MINUTES function, the function returns the integer number of minutes between them.

• **Syntax 3**  If you pass a TIMESTAMP value and an INTEGER value to the MINUTES function, the function returns the TIMESTAMP result of adding the integer number of minutes to *timestamp-expression* argument. Similarly, if the first argument to MINUTES is a TIME value, then the result is also a TIME value. Syntax 3 does not support implicit conversion of the first argument. It may be necessary to explicitly cast the first argument to a DATE, TIME or TIMESTAMP value. If the first argument is of type DATE, midnight is assumed for the time portion.

Since MINUTES returns an integer, overflow can occur when Syntax 1 is used with TIMESTAMP values greater than or equal to 4083-03-23 02:08:00.

Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.

**See also**

- “DATEDIFF function [Date and time]” on page 205
- “DATEADD function [Date and time]” on page 204
- “CAST function [Data type conversion]” on page 174

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

**Example**

The following statements return the value 240, signifying that the second TIMESTAMP value is 240 minutes after the first. It is recommended that you use the second example (DATEDIFF).

```sql
SELECT MINUTES( '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
SELECT DATEDIFF( minute, '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
```

The following statement returns the value 1051040527:

```sql
SELECT MINUTES( '1998-07-13 06:07:12' );
```

The following statements return the TIMESTAMP value 1999-05-12 21:10:07.000. The first statement requires an explicit cast of the literal string parameter. It is recommended that you use the second example (DATEADD).

```sql
SELECT MINUTES( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5);
SELECT DATEADD( minute, 5, '1999-05-12 21:05:07' );
```
The following statement returns 'TIME', illustrating that the MINUTES function returns a TIME value when it is called with a TIME argument:

```
SELECT EXPRTYPE('SELECT MINUTES( CAST( ''13:45:00.000'' AS TIME ), 16 )', 1);
```

**MOD function [Numeric]**

Returns the remainder when one whole number is divided by another.

**Syntax**

```
MOD( dividend, divisor )
```

**Parameters**

- **dividend** The dividend, or numerator of the division.
- **divisor** The divisor, or denominator of the division.

**Returns**

- SMALLINT
- INT
- NUMERIC

**Remarks**

Division involving a negative dividend gives a negative or zero result. The sign of the divisor has no effect.

**See also**

- “REMAINDER function [Numeric]” on page 344

**Standards and compatibility**

- SQL/2008 The MOD function is part of optional SQL/2008 language feature T441.

**Example**

The following statement returns the value 2:

```
SELECT MOD( 5, 3 );
```

**MONTH function [Date and time]**

Returns the month of the given date.

**Syntax**

```
MONTH( date-expression )
```
Parameters
● date-expression A value of type DATE.

Returns
SMALLINT

Remarks
The value returned is a number between 1 and 12, corresponding to the month of the given date.

Standards and compatibility
● SQL/2008 Vendor extension.

Example
The following statement returns the value 7:

```
SELECT MONTH( '1998-07-13' );
```

MONTHNAME function [Date and time]
Returns the name of the month from a date.

Syntax
```
MONTHNAME( date-expression )
```

Parameters
● timestamp-expression A TIMESTAMP value.

Returns
VARCHAR

Remarks
The MONTHNAME function returns a string, even if the result is numeric, such as 2 for the month of February.

See also
● “DATEPART function [Date and time]” on page 208

Standards and compatibility
● SQL/2008 Vendor extension.

Example
The following statement returns the value September:

```
SELECT MONTHNAME( '1998-09-05' );
```
MONTHS function [Date and time]

Manipulates a TIMESTAMP or returns the number of months between two TIMESTAMP values. See the Remarks section below.

Syntax 1
MONTHS( timestamp-expression )

Syntax 2
MONTHS( timestamp-expression, timestamp-expression )

Syntax 3
MONTHS( timestamp-expression, integer-expression )

Parameters

- **timestamp-expression** A date and time of type TIMESTAMP.
- **integer-expression** The integer number of months (of type SMALLINT) to be added to the timestamp-expression. If integer-expression is negative, the appropriate number of months is subtracted from timestamp-expression. If you supply an integer-expression, the timestamp-expression must be explicitly cast as a TIME, DATE or TIMESTAMP data type. If timestamp-expression is a TIME value, the current month is assumed.

For information about casting data types, see “CAST function [Data type conversion]” on page 174.

Returns

INTEGER with Syntax 1 or Syntax 2.

TIMESTAMP with Syntax 3.

Remarks

The result of the MONTHS function depends on its arguments. The MONTHS function ignores hours, minutes, and seconds in its arguments.

- **Syntax 1** If you pass a single timestamp-expression to the MONTHS function, it will return the number of months between 0000-02 and timestamp-expression as an INTEGER.

```
Note
0000-02 is not meant to imply an actual date; it is the default date used by the MONTHS function.
```

- **Syntax 2** If you pass two TIMESTAMP values to the MONTHS function, the function returns the integer number of months between them.

- **Syntax 3** If you pass a TIMESTAMP value and a SMALLINT value to the MONTHS function, the function returns the TIMESTAMP result of adding the integer number of months to timestamp-expression.

Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.
The value of MONTHS is calculated from the number of first days of the month between the two dates.

See also
- “DATEDIFF function [Date and time]” on page 205
- “DATEADD function [Date and time]” on page 204

Standards and compatibility
- SQL/2008 Vendor extension.

Example

The following statements return the value 2, signifying that the second date is two months after the first. It is recommended that you use the second example (DATEDIFF).

```
SELECT MONTHS( '1999-07-13 06:07:12', '1999-09-13 10:07:12' );
```

```
SELECT DATEDIFF( month,
'1999-07-13 06:07:12',
'1999-09-13 10:07:12' ) ;
```

The following statement returns the value 23981:

```
SELECT MONTHS( '1998-07-13 06:07:12');
```

The following statements return the TIMESTAMP value 1999-10-12 21:05:07.000. It is recommended that you use the second example (DATEADD).

```
SELECT MONTHS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5);
```

```
SELECT DATEADD( month, 5, '1999-05-12 21:05:07' );
```

**NCHAR function [String]**

Returns an NCHAR string containing one character whose Unicode code point is given in the parameter, or NULL if the value is not a valid code point value.

**Syntax**

```
NCHAR( integer )
```

**Parameters**

- **integer** The number to be converted to the corresponding Unicode code point.

**Returns**

NVARCHAR
See also

- “CONNECTION_EXTENDED_PROPERTY function [String]” on page 184
- “TO_NCHAR function [String]” on page 387
- “TO_CHAR function [String]” on page 386
- “UNICODE function [String]” on page 397
- “UNISTR function [String]” on page 398

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example returns the ALEF Arabic letter, which is Unicode code point U+627:

```
SELECT NCHAR( 1575 );
```

**NEWID function [Miscellaneous]**

Generates a UUID (Universally Unique Identifier) value. A UUID is the same as a GUID (Globally Unique Identifier).

Syntax

```
NEWID()
```

Parameters

There are no parameters associated with the NEWID function.

Returns

UNIQUEIDENTIFIER

Remarks

The NEWID function can be used in a DEFAULT clause for a column.

UUIDs can be used to uniquely identify rows in a table. A value produced on one computer does not match a value produced on another computer, so they can be used as keys in synchronization and replication environments.

UUIDs contain hyphens for compatibility with other RDBMSs. You change this by setting the `uuid_has_hyphens` option to Off. For more information, see “uuid_has_hyphens option” [SQL Anywhere Server - Database Administration].

The NEWID function is non-deterministic; successive calls will return different values. The query optimizer does not cache the results of the NEWID function.

For more information about non-deterministic functions, see “Function caching” [SQL Anywhere Server - SQL Usage].
See also

- “The NEWID default” [SQL Anywhere Server - SQL Usage]
- “STRTOUUID function [String]” on page 377
- “UUIDTOSTR function [String]” on page 401
- “UNIQUEIDENTIFIER data type” on page 126

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement creates a table named mytab with two columns. Column pk has a unique identifier data type, and assigns the NEWID function as the default value. Column c1 has an integer data type.

```sql
CREATE TABLE mytab(
    pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    c1 INT );
```

The following statement returns a unique identifier as a string:

```sql
SELECT UUIDTOSTR( NEWID() );
```

For example, the value returned might be 96603324-6FF6-49DE-BF7D-F44C1C7E6856.

**NEXT_CONNECTION function [System]**

Returns an identifying number for the next connection.

Syntax

```sql
NEXT_CONNECTION( connection-id [, database-id ] )
```

Returns

INT

Parameters

- **connection-id**  An integer, usually returned from a previous call to NEXT_CONNECTION. If connection-id is NULL, NEXT_CONNECTION returns the most recent connection ID.

- **database-id**  An integer representing one of the databases on the current server. If you supply no database-id, the current database is used. If you supply NULL, then NEXT_CONNECTION returns the next connection regardless of database.

Remarks

NEXT_CONNECTION can be used to enumerate the connections to a database. Connection IDs are generally created in monotonically increasing order. This function returns the next connection ID in reverse order.
To get the connection ID value for the most recent connection, enter NULL as the `connection-id`. To get the subsequent connection, enter the previous return value. The function returns NULL when there are no more connections in the order.

`NEXT_CONNECTION` is useful for disconnecting all the connections created before a specific time. However, because `NEXT_CONNECTION` returns the connection IDs in reverse order, connections made after the function is started are not returned. To ensure that all connections are disconnected, prevent new connections from being created before you run `NEXT_CONNECTION`.

*Cloud note:* Because of tenant database isolation rules, when this function is called in the cloud it returns only information about the current tenant database.

**Privileges**

No privileges are required to execute this function to return the current connection ID. To execute this function for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

**Example**

The following statement returns an identifier as an integer value for the first connection on the current database:

```sql
SELECT NEXT_CONNECTION( NULL );
```

The following statement returns an integer value like 5.

```sql
SELECT NEXT_CONNECTION( 10 );
```

The following call returns the next connection ID in reverse order from the specified `connection-id` on the current database:

```sql
SELECT NEXT_CONNECTION( connection-id );
```

The following call returns the next connection ID in reverse order from the specified `connection-id`, regardless of database:

```sql
SELECT NEXT_CONNECTION( connection-id, NULL );
```

The following call returns the next connection ID in reverse order from the specified `connection-id` on the specified database:

```sql
SELECT NEXT_CONNECTION( connection-id, database-id );
```

The following call returns the first or earliest connection, regardless of database:

```sql
SELECT NEXT_CONNECTION( NULL, NULL );
```

The following call returns the first or earliest connection on the specified database:

```sql
SELECT NEXT_CONNECTION( NULL, database-id );
```
NEXT_DATABASE function [System]

Returns an identifying number for a database.

Syntax

NEXT_DATABASE( database-id )

Parameters

● database-id An integer that specifies the ID number of the database.

Returns

INT

Remarks

The NEXT_DATABASE function is used to enumerate the databases running on a database server. To get the first database specify NULL; to get each subsequent database, specify the previous return value. The function returns NULL when there are no more databases. The database ID numbers are not returned in a particular order, but you can tell the order in which databases were started on the server using the database ID. The first database started on the server is assigned the value 0, and for subsequent databases started on the server, the database IDs are incremented by 1.

* Cloud note: Because of tenant database isolation rules, when this function is called in the cloud it returns only information about the current tenant database.

Privileges

No privileges are required to execute this function to return the current database. To execute this function for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

See also

● “DB_NAME function [System]” on page 217
● “sa_db_list system procedure” on page 1119

Standards and compatibility

● SQL/2008 Vendor extension.

Example

The following statement returns the value 0, the first database value:

SELECT NEXT_DATABASE( NULL );

The following statement returns NULL, only when one database has been started:

SELECT NEXT_DATABASE( 0 );

NEXT_HTTP_HEADER function [Web service]

Returns the next HTTP header name.
Syntax

NEXT_HTTP_HEADER( header-name )

Parameters

- **header-name**  The name of the previous request header. If header-name is NULL, this function returns the name of the first HTTP request header.

Returns

LONG VARCHAR.

**Note**
The result data type is a LONG VARCHAR. If you use NEXT_HTTP_HEADER in a SELECT INTO statement, you must have an Unstructured Data Analytics Option license or use CAST and set HTML_DECODE to the correct data type and size.

Remarks

This function is used to iterate over the HTTP request headers returning the next HTTP header name. Calling it with NULL causes it to return the name of the first header. Subsequent headers are retrieved by passing the name of the previous header to the function. This function returns NULL when called with the name of the last header, or when not called from a web service.

Calling this function repeatedly returns all the header fields exactly once, but not necessarily in the order they appear in the HTTP request.

See also

- “HTTP_HEADER function [Web service]” on page 266
- “sa_http_header_info system procedure” on page 1154
- “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement displays the name and values of the HTTP request headers in the database server messages window when used within a stored procedure that is called by an HTTP web service:

```sql
BEGIN
    declare header_name long varchar;
    declare header_value long varchar;
    set header_name = NULL;
    header_loop:
    LOOP
        SET header_name = NEXT_HTTP_HEADER( header_name );
        IF header_name IS NULL THEN
            LEAVE header_loop
        END IF;
```
NEXT_HTTP_RESPONSE_HEADER function [Web service]

Returns the next HTTP response header name.

Syntax

NEXT_HTTP_RESPONSE_HEADER( header-name )

Parameters

- header-name The name of the previous response header. If header-name is NULL, this function returns the name of the first HTTP response header.

Returns

LONG VARCHAR

Remarks

This function is used to iterate over the HTTP response headers returning the next HTTP response header name. Calling it with NULL causes it to return the name of the first response header. Subsequent response headers are retrieved by passing the name of the previous response header to the function. This function returns NULL when called with the name of the last response header, or if it is not called from a web service.

Calling this function repeatedly returns all the response header fields exactly once, but not necessarily in the order they appear in the HTTP response.

See also

- “HTTP_RESPONSE_HEADER function [Web service]” on page 268
- “HTTP request header management” [SQL Anywhere Server - Programming]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement displays the name and values of the HTTP response headers in the database server messages window when used within a stored procedure that is called by an HTTP web service:

```
BEGIN
    declare header_name long varchar;
    declare header_value long varchar;
    set header_name = NULL;
    header_loop:
        LOOP
            SET header_value = HTTP_HEADER( header_name );
            MESSAGE 'HEADER: ', header_name, '=',
                header_value TO CONSOLE;
        END LOOP;
END;
```
SET header_name = NEXT_HTTP_RESPONSE_HEADER( header_name )
IF header_name IS NULL THEN
  LEAVE header_loop
END IF;
SET header_value = HTTP_RESPONSE_HEADER( header_name )
MESSAGE 'RESPONSE HEADER: ', header_name, ' = ', header_value TO CONSOLE;
END LOOP;

NEXT_HTTP_VARIABLE function [Web service]

Returns the next HTTP variable name.

Syntax
NEXT_HTTP_VARIABLE( var-name )

Parameters
- var-name  The name of the previous variable. If var-name is NULL, this function returns the name of the first HTTP variable.

Returns
LONG VARCHAR.

Remarks
This function iterates over the HTTP variables included within a request. Calling it with NULL causes it to return the name of the first variable. Subsequent variables are retrieved by passing the function the name of the previous variable. This function returns NULL when called with the name of the final variable or when not called from a web service.

Calling this function repeatedly returns all the variables exactly once, but not necessarily in the order they appear in the HTTP request. The variables url or url1, url2, ..., url10 are included if URL PATH is set to ON or ELEMENTS, respectively.

See also
- “HTTP_VARIABLE function [Web service]” on page 270
- “NEXT_HTTP_HEADER function [Web service]” on page 306
- “sa_http_variable_info system procedure” on page 1158
- “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement returns the name of the first HTTP variable when used within a stored procedure that is called by an HTTP web service:
BEGIN
    DECLARE variable_name LONG VARCHAR;
    DECLARE variable_value LONG VARCHAR;
    SET variable_name = NULL;
    SET variable_name = NEXT_HTTP_VARIABLE( variable_name );
    SET variable_value = HTTP_VARIABLE( variable_name );
END;

**NEXT_SOAP_HEADER function [SOAP]**

Returns the next header key in a SOAP request header.

**Syntax**

```
NEXT_SOAP_HEADER( header-key )
```

**Parameters**

- **header-key**  The XML local name of the top level XML element for the given header entry.

**Returns**

LONG VARCHAR

**Remarks**

If you specify NULL for the header-key, the function returns the header key for the first header entry found in the SOAP header.

This function returns NULL if called with the last header-key.

**See also**

- “SOAP_HEADER function [SOAP]” on page 363
- “Tutorial: Using SQL Anywhere to access a SOAP/DISH service” [SQL Anywhere Server - Programming]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

**Example**

The following statement processes all the keys located in the SOAP request header when used within a stored procedure that is called by an HTTP web service. When it processes the Authentication key, it also obtains the key's value.

```
BEGIN
    DECLARE hd_key LONG VARCHAR;
    DECLARE hd_entry LONG VARCHAR;
    header_loop:
    LOOP
        SET hd_key = NEXT_SOAP_HEADER( hd_key );
        IF hd_key IS NULL THEN
            -- no more header entries
```

310 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
LEAVE header_loop;
END IF;
IF hd_key = 'Authentication' THEN
SET hd_entry = SOAP_HEADER( hd_key );
END IF;
END LOOP header_loop;
END;

NOW function [Date and time]

Returns the current date and time as a TIMESTAMP value. The accuracy is limited by the accuracy of the system clock.

Syntax

NOW([ * ])

Returns

TIMESTAMP

Remarks

NOW is equivalent to the GETDATE function and the CURRENT TIMESTAMP special value. NOW(*) and NOW() are equivalent constructions.

Each instance of the NOW function in a request is evaluated at most once. Multiple instances of NOW in the same request may or may not share the identical TIMESTAMP value.

See also

- “CURRENT TIME special value” on page 67
- “CURRENT TIMESTAMP special value” on page 68
- “CURRENT UTC TIMESTAMP special value” on page 70
- “DATE data type” on page 115
- “DATE function [Date and time]” on page 203
- “DATETIME data type” on page 116
- “DATETIME function [Date and time]” on page 209
- “DATETIMEOFFSET data type” on page 117
- “Expressions” on page 21
- “GETDATE function [Date and time]” on page 251
- “ISDATE function [Data type conversion]” on page 275
- “SMALLDATETIME data type” on page 119
- “TIME data type” on page 120
- “TIMESTAMP special value” on page 76
- “TIMESTAMP data type” on page 121
- “UTC TIMESTAMP special value” on page 78

Standards and compatibility

- SQL/2008 Vendor extension.
Example
The following statement returns the current date and time:

```
SELECT NOW( * ) ;
```

NULLIF function [Miscellaneous]
Provides an abbreviated CASE expression by comparing expressions.

Syntax
```
NULLIF( expression-1, expression-2 )
```

Parameters
- `expression-1` An expression to be compared.
- `expression-2` An expression to be compared.

Returns
Data type of the first argument.

Remarks
NULLIF compares the values of the two expressions.

If the first expression equals the second expression, NULLIF returns NULL.

If the first expression does not equal the second expression, or if the second expression is NULL, NULLIF returns the first expression.

The NULLIF function provides a short way to write some CASE expressions.

See also
- “CASE expressions” on page 24

Standards and compatibility
- SQL/2008 Core feature.

Example
The following statement returns the value a:

```
SELECT NULLIF( 'a', 'b' ) ;
```

The following statement returns NULL:

```
SELECT NULLIF( 'a', 'a' ) ;
```
NUMBER function [Miscellaneous]

Generates numbers starting at 1 for each successive row in the results of the query. The NUMBER function is primarily intended for use in SELECT lists.

Due to limitations imposed by the NUMBER function (described in the Remarks section below), use the ROW_NUMBER function instead. The ROW_NUMBER function provides the same functionality, but without the limitations of the NUMBER function. See “ROW_NUMBER function [Miscellaneous]” on page 353.

Syntax

NUMBER([ * ])

Returns

INT

Remarks

You can use NUMBER(*) in a SELECT list to provide a sequential numbering of the rows in the result set. NUMBER(*) returns the value of the ANSI row number of each result row. The NUMBER function can return positive or negative values, depending on how the application scrolls through the result set. For insensitive cursors, the value of NUMBER(*) will always be positive because the entire result set is materialized at OPEN.

In addition, the row number may be subject to change for some cursor types. The value is fixed for insensitive cursors and scroll cursors. If there are concurrent updates, it may change for dynamic and sensitive cursors.

A syntax error is generated if you use the NUMBER function in: a DELETE statement, a WHERE clause, a HAVING clause, an ORDER BY clause, a subquery, a query involving aggregation, any constraint, a GROUP BY clause, a DISTINCT clause, a set operator (UNION, EXCEPT, INTERSECT), or a derived table.

NUMBER(*) can be used in a view (subject to the above restrictions), but the view column corresponding to the expression involving NUMBER(*) can be referenced at most once in the query or outer view, and the view cannot participate as a NULL-supplying table in a left outer join or full outer join.

In embedded SQL, care should be exercised when using a cursor that references a query containing a NUMBER(*) function. In particular, this function returns negative numbers when a database cursor is positioned using relative to the end of the cursor (an absolute position with a negative offset).

You can use NUMBER in the right side of an assignment in the SET clause of an UPDATE statement. For example, \texttt{SET x = NUMBER(*)}.

The NUMBER function can also be used to generate primary keys when using the INSERT from SELECT statement, although using an AUTOINCREMENT clause is a preferred mechanism for generating sequential primary keys.

For information about the AUTOINCREMENT clause, see “CREATE TABLE statement” on page 690.
NUMBER(*) and NUMBER() are semantically equivalent.

See also
- “INSERT statement” on page 860

Standards and compatibility
- SQL/2008   Vendor extension.

Example
The following statement returns a sequentially-numbered list of departments:

```sql
SELECT NUMBER( * ), DepartmentName
FROM GROUPO.Departments
WHERE DepartmentID > 5
ORDER BY DepartmentName;
```

**PATINDEX function [String]**
Returns an integer representing the starting position of the first occurrence of a pattern in a string.

Syntax
```
PATINDEX( '%pattern%', string-expression )
```

Parameters
- **pattern**  The pattern to be searched for. If the leading percent wildcard is omitted, the PATINDEX function returns one (1) if the pattern occurs at the beginning of the string, and zero if it does not.

The pattern uses the same wildcards as the LIKE comparison. These are as follows:

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ (underscore)</td>
<td>Any one character</td>
</tr>
<tr>
<td>% (percent)</td>
<td>Any string of zero or more characters</td>
</tr>
<tr>
<td>[ ]</td>
<td>Any single character in the specified range or set</td>
</tr>
<tr>
<td>[^]</td>
<td>Any single character not in the specified range or set</td>
</tr>
</tbody>
</table>

- **string-expression**  The string to be searched for the pattern.

Returns
INT

Remarks
The PATINDEX function returns the starting position of the first occurrence of the pattern. If the pattern is not found, it returns zero (0).
See also

- “LIKE search condition” on page 47
- “LOCATE function [String]” on page 287
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the value 2:

```
SELECT PATINDEX( '%hoco%', 'chocolate' );
```

The following statement returns the value 11:

```
SELECT PATINDEX( '%4_5_%', '0a1A 2a3A 4a5A' );
```

The following statement returns 14 which is the first non-alphanumeric character in the string expression. The pattern '%[^a-z0-9]%' can be used instead of '%[^a-zA-Z0-9]%' if the database is case insensitive.

```
SELECT PATINDEX( '%[^a-zA-Z0-9]%', 'SQLAnywhere16 has many new features' );
```

The following statement can be used to retrieve everything up to and including the first non-alphanumeric character in a string:

```
SELECT LEFT( @string, PATINDEX( '%[^a-zA-Z0-9]%', @string ) );
```

The following statements create a table, myTable, and populate it with various strings containing alphanumeric characters, spaces (blanks), and non-alphanumeric characters. Then, the SELECT statement and subsequent results show how you can use PATINDEX to find the starting position of spaces and non-alphanumeric characters in the strings:

```
CREATE TABLE myTable( col1 LONG VARCHAR );

INSERT INTO myTable (col1) VALUES( 'the quick brown fox jumped over the lazy dog' ),
( 'the quick brown fox $$$ jumped over the lazy dog' ),
( 'the quick brown fox 0999 jumped over the lazy dog' ),
( 'the quick brown fox ** jumped over the lazy dog' ),
( 'thequickbrownfoxjumpedoverthelazydog' ),
( 'thequickbrownfoxjum999pedoovertheazydog' ),
( 'thequick$$$$brownfox' ),
( 'the quick brown fox$$ jumped over the lazy dog' );

SELECT col1,
//position of first non-alphanumeric character or space:
    PATINDEX( '%[^a-z0-9]%', col1 ) AS blank_posn,
//position of first non-alphanumeric char that isn't a space:
    PATINDEX( '%[^ a-zA-Z0-9]%', col1 ) AS non_alpha_char,
//everything up to and including first non-alphanumeric char that isn't a space:
    LEFT ( col1, PATINDEX( '%[^ a-zA-Z0-9]%', col1 ) ) AS left_str,
//first non-alphanumeric char that isn't a space, and everything to the right:
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 315
SUBSTRING ( col1, PATINDEX( '^[^ a-zA-Z0-9]%', col1 ) ) AS sub_str
FROM myTable;

<table>
<thead>
<tr>
<th>col1</th>
<th>blank_posn</th>
<th>non_alpha_char</th>
<th>left_str</th>
<th>sub_str</th>
</tr>
</thead>
<tbody>
<tr>
<td>the quick brown fox jumped over the lazy dog</td>
<td>4</td>
<td>0</td>
<td></td>
<td>the quick brown fox jumped over the lazy dog</td>
</tr>
<tr>
<td>the quick brown fox $$$$$ jumped over the lazy dog</td>
<td>4</td>
<td>21</td>
<td></td>
<td>$$$$$ jumped over the lazy dog</td>
</tr>
<tr>
<td>the quick brown fox 0999 jumped over the lazy dog</td>
<td>4</td>
<td>0</td>
<td></td>
<td>the quick brown fox 0999 jumped over the lazy dog</td>
</tr>
<tr>
<td>the quick brown fox ** jumped over the lazy dog</td>
<td>4</td>
<td>21</td>
<td></td>
<td>** jumped over the lazy dog</td>
</tr>
<tr>
<td>thequickbrownfoxjumpe-doverthelazydog</td>
<td>0</td>
<td>0</td>
<td></td>
<td>thequickbrownfoxjumpe-doverthelazydog</td>
</tr>
<tr>
<td>thequickbrownfox-jum999pedoverthelazydog</td>
<td>0</td>
<td>0</td>
<td></td>
<td>thequickbrownfox-jum999pedoverthelazydog</td>
</tr>
<tr>
<td>thequick$$$$brownfox</td>
<td>9</td>
<td>9</td>
<td></td>
<td>thequick$ $$$$$brownfox</td>
</tr>
<tr>
<td>the quick brown fox$$ jumped over the lazy dog</td>
<td>4</td>
<td>20</td>
<td></td>
<td>$$ jumped over the lazy dog</td>
</tr>
</tbody>
</table>

**PERCENT_RANK function [Ranking]**

For any row X, defined by the function's arguments and ORDER BY specification, the PERCENT_RANK function determines the rank of row X - 1, divided by the number of rows in the group.

**Syntax**

```sql
PERCENT_RANK( ) OVER ( window-spec )
```

*window-spec*: see the Remarks section below

**Returns**

The PERCENT_RANK function returns a DOUBLE value between 0 and 1.
Remarks

Elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. When used as a window function, you must specify an ORDER BY clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE clause. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also

- “WINDOW clause” on page 1051
- “CUME_DIST function [Ranking]” on page 200
- “DENSE_RANK function [Ranking]” on page 222
- “RANK function [Ranking]” on page 326

Standards and compatibility

- **SQL/2008**  PERCENT_RANK is part of optional SQL/2008 language feature T612, ”Advanced OLAP operations”.

Example

The following example returns a result set that shows the ranking of New York employees' salaries in descending order by gender:

```sql
SELECT DepartmentID, Surname, Salary, Sex,
PERCENT_RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) "Rank"
FROM GROUPO.Employees
WHERE State IN ('NY');
```

<table>
<thead>
<tr>
<th>DepartmentID</th>
<th>Surname</th>
<th>Salary</th>
<th>Sex</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Martel</td>
<td>55700.000</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>Guevara</td>
<td>42998.000</td>
<td>M</td>
<td>0.333333333</td>
</tr>
<tr>
<td>100</td>
<td>Soo</td>
<td>39075.000</td>
<td>M</td>
<td>0.666666667</td>
</tr>
<tr>
<td>400</td>
<td>Ahmed</td>
<td>34992.000</td>
<td>M</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>Davidson</td>
<td>57090.000</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td>400</td>
<td>Blaikie</td>
<td>54900.000</td>
<td>F</td>
<td>0.333333333</td>
</tr>
<tr>
<td>100</td>
<td>Whitney</td>
<td>45700.000</td>
<td>F</td>
<td>0.666666667</td>
</tr>
<tr>
<td>400</td>
<td>Wetherby</td>
<td>35745.000</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>
**PI function [Numeric]**

Returns the numeric value PI.

**Syntax**

\[
\text{PI}\left( \ast \right)
\]

**Returns**

DOUBLE

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Remarks**

This function returns a DOUBLE value.

PI(*) and PI() are semantically equivalent.

**Example**

The following statement returns the value 3.141592653(...):

\[
\text{SELECT PI}\left( \ast \right);
\]

**PLAN function [Miscellaneous]**

Returns the long plan optimization strategy of a SQL statement, as a string.

**Syntax**

\[
\text{PLAN}\left( \text{string-expression}, \left[ \text{cursor-type} \left[ \text{update-status} \right] \right] \right)
\]

**Parameters**

- **string-expression** The SQL statement, which is commonly a SELECT statement but which may also be an UPDATE, MERGE, or DELETE statement.

- **cursor-type** A string. cursor-type can be asensitive (default), insensitive, sensitive, or keyset-driven.

- **update-status** A string parameter accepting one of the following values indicating how the optimizer should treat the given cursor:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ-ONLY</td>
<td>The cursor is read-only.</td>
</tr>
<tr>
<td>READ-WRITE (default)</td>
<td>The cursor can be read or written to.</td>
</tr>
</tbody>
</table>
Value | Description
--- | ---
FOR UPDATE | The cursor can be read or written to. This is exactly the same as READ-WRITE.

Returns

LONG VARCHAR

See also

- “Advanced: Query execution plans” [SQL Anywhere Server - SQL Usage]
- “EXPLANATION function [Miscellaneous]” on page 244
- “GRAPHICAL_PLAN function [Miscellaneous]” on page 252

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement passes a SELECT statement as a string parameter and returns the plan for executing the query:

```sql
SELECT PLAN('SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100');
```

This information can help with decisions about indexes to add or how to structure your database for better performance.

The following statement returns a string containing the text plan for an INSENSITIVE cursor over the query `SELECT * FROM Departments WHERE DepartmentID > 100;`:

```sql
SELECT PLAN('SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100', 'insensitive', 'read-only');
```

---

**POWER function [Numeric]**

Calculates one number raised to the power of another.

**Syntax**

```
POWER( numeric-expression-1, numeric-expression-2 )
```

**Parameters**

- `numeric-expression-1` The base.
- `numeric-expression-2` The exponent.

**Returns**

`DOUBLE`
Remarks
This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If any argument is NULL, the result is a NULL value.

Standards and compatibility

- **SQL/2008** The POWER function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example
The following statement returns the value 64:

```
SELECT POWER( 2, 6 );
```

PROPERTY function [System]
Returns the value of the specified database server property as a string.

Syntax
```
PROPERTY( { property-id | property-name } [, second-parameter ] )
```

Parameters

- **property-id** An integer that is the property-number of the database server property. This number can be determined from the PROPERTY_NUMBER function. The property-id is commonly used when looping through a set of properties.

- **property-name** A string giving the name of the database property.

- **second-parameter** You can specify a second parameter for some properties, as follows:

<table>
<thead>
<tr>
<th>Property</th>
<th>Second parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventType-Desc</td>
<td>positive-integer</td>
<td>Specify an event ID to return the event type description. See “EventTypeDesc server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>EventType-Name</td>
<td>positive-integer</td>
<td>Specify an event ID to return the event type name. See “EventTypeName server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Function-MaxParms</td>
<td>positive-integer</td>
<td>Specify a function number to return the maximum number of parameters that can be specified for the function. See “Function-MaxParms server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Property</td>
<td>Second parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Function-MinParms</td>
<td><em>positive-integer</em></td>
<td>Specify a function number to return the minimum number of parameters that must be specified for the function. See “Function-MinParms server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Function-Name</td>
<td><em>positive-integer</em></td>
<td>Specify a function number to return the function name. See “FunctionName server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Message</td>
<td><em>positive-integer</em></td>
<td>Specify a line number to return the contents of the corresponding line in the database server messages window, prefixed by the date and time the message appeared. See “Message server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Message-Text</td>
<td><em>positive-integer</em></td>
<td>Specify a line number to return the text associated with the specified line number in the database server messages window, without a date and time prefix. See “MessageText server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Message-Time</td>
<td><em>positive-integer</em></td>
<td>Specify a line number to return the date and time associated with the specified line number in the database server messages window. See “MessageTime server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>RemoteCapability</td>
<td><em>positive-integer</em></td>
<td>Specify a remote capability ID to return the remote capability name associated with the ID. See “RemoteCapability server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
</tbody>
</table>

**Returns**

VARCHAR, LONG VARCHAR

**Remarks**

Each property has both a number and a name, but the number is subject to change between releases, and should not be used as a reliable identifier for a given property.

**See also**

- “List of database server properties” [SQL Anywhere Server - Database Administration]
- “DB_PROPERTY function [System]” on page 218

**Standards and compatibility**

- SQL/2008 Vendor extension.
Example

The following statement returns the name of the current database server:

```
SELECT PROPERTY( 'Name' );
```

**PROPERTY_DESCRIPTION function [System]**

Returns a description of a property.

**Syntax**

```
PROPERTY_DESCRIPTION( { property-id | property-name } )
```

**Parameters**

- **property-id**  
  An integer that is the property-number of the database property. This number can be determined from the PROPERTY_NUMBER function. The property-id is commonly used when looping through a set of properties.

- **property-name**  
  A string giving the name of the database property.

**Returns**

VARCHAR

**Remarks**

Each property has both a number and a name, but the number is subject to change between releases, and should not be used as a reliable identifier for a given property.

**See also**

- “Connection, database, and database server properties” [SQL Anywhere Server - Database Administration]

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statement returns Number of index insertions, the description of the IndAdd property:

```
SELECT PROPERTY_DESCRIPTION( 'IndAdd' );
```

**PROPERTY_NAME function [System]**

Returns the name of the property with the supplied property ID for the specified connection level.

**Syntax**

```
PROPERTY_NAME( property-id [, property-scope ] )
```
property-scope:
NULL
'server'
'database'
'db'
'connection'
'conn'

Parameters

- **property-id**  The property ID of the database property.
- **property-scope**  The scope of the property, or NULL.

Returns

VARCHAR

See also

- “List of connection properties” [SQL Anywhere Server - Database Administration]
- “List of database server properties” [SQL Anywhere Server - Database Administration]
- “List of database properties” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the server-level property associated with property ID 102:

```
SELECT PROPERTY_NAME( 102, 'server' );
```

**PROPERTY_NUMBER function [System]**

Returns the property number of the property with the supplied property-name.

Syntax

```
PROPERTY_NUMBER( property-name )
```

Parameters

- **property-name**  A property name.

Returns

INT

Remarks

Each property has both a number and a name, but the number is subject to change between releases, and should not be used as a reliable identifier for a given property. When either property number or property name can be used, it is preferable to use the property name. Always use the PROPERTY_NUMBER function to ensure that the property number is current for the server being used.
See also
- “Connection, database, and database server properties” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement returns the property number of the PAGESIZE property as an integer:

```
SELECT PROPERTY_NUMBER('PAGESIZE');
```

QUARTER function [Date and time]
Returns a number indicating the quarter of the year from the supplied TIMESTAMP expression.

Syntax
```
QUARTER( timestamp-expression )
```

Parameters
- **timestamp-expression**  The date you want the quarter for.

Returns
INTEGER

Remarks
The quarters are as follows:

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Period (inclusive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>January 1 to March 31</td>
</tr>
<tr>
<td>2</td>
<td>April 1 to June 30</td>
</tr>
<tr>
<td>3</td>
<td>July 1 to September 30</td>
</tr>
<tr>
<td>4</td>
<td>October 1 to December 31</td>
</tr>
</tbody>
</table>

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement returns the value 2:

```
SELECT QUARTER('1987/05/02');
```
RADIANS function [Numeric]

Converts a number from degrees to radians.

Syntax
RADIANS( numeric-expression )

Parameters
- numeric-expression A number, in degrees. This angle is converted to radians.

Returns
DOUBLE

Remarks
This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement returns a value of approximately 0.5236:

```
SELECT RADIANS( 30 );
```

RAND function [Numeric]

Returns a random number in the interval 0 to 1, with an optional seed.

Syntax
RAND( [integer-expression] )

Parameters
- integer-expression An optional seed used to create a random number. This argument allows you to create repeatable random number sequences.

Returns
DOUBLE

Remarks
The RAND function is a multiplicative linear congruential random number generator. See Park and Miller (1988), CACM 31(10), pp. 1192-1201 and Press et al. (1992), Numerical Recipes in C (2nd edition, Chapter 7, pp. 279). The result of calling the RAND function is a pseudo-random number \( n \) where \( 0 < n < 1 \) (neither 0.0 nor 1.0 can be the result).
When a connection is made to the server, the random number generator seeds an initial value. Each connection is uniquely seeded so that it sees a different random sequence from other connections. You can also specify a seed value (integer-expression) as an argument. Normally, you should only do this once before requesting a sequence of random numbers through successive calls to the RAND function. If you initialize the seed value more than once, the sequence is restarted. If you specify the same seed value, the same sequence is generated. Seed values that are close in value generate similar initial sequences, with divergence further out in the sequence.

Never combine the sequence generated from one seed value with the sequence generated from a second seed value, in an attempt to obtain statistically random results. In other words, do not reset the seed value at any time during the generation of a sequence of random values.

The RAND function is treated as a non-deterministic function. The query optimizer does not cache the results of the RAND function.

For more information about non-deterministic functions, see “Function caching” [SQL Anywhere Server - SQL Usage].

Standards and compatibility

- SQL2008  Vendor extension.

Example

The following statements produce eleven random results. Each subsequent call to the RAND function where a seed is not specified continues to produce different results:

```
SELECT RAND(1);
SELECT RAND(), RAND(), RAND(), RAND(), RAND();
SELECT RAND(), RAND(), RAND(), RAND(), RAND();
```

The following statement produces two sets of results with identical sequences, since the seed value is specified twice:

```
SELECT RAND(1), RAND(), RAND(), RAND(), RAND();
SELECT RAND(1), RAND(), RAND(), RAND(), RAND();
```

The following example produces five results that are near each other in value, and do not have a random distribution. For this reason, calling the RAND function more than once with similar seed values is not recommended:

```
SELECT RAND(1), RAND(2), RAND(3), RAND(4), RAND(5);
```

The following example produces five identical results, and should be avoided:

```
SELECT RAND(1), RAND(1), RAND(1), RAND(1), RAND(1);
```

RANK function [Ranking]

Calculates the value of a rank in a group of values. For ties, the RANK function leaves a gap in the ranking sequence.
Syntax

\texttt{RANK( ) OVER ( \textit{window-spec} )}

\textit{window-spec} : see the Remarks section below

Returns

INTEGER

Remarks

Elements of \textit{window-spec} can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. When used as a window function, you must specify an ORDER BY clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE clause. See the \textit{window-spec} definition for the WINDOW clause. See “WINDOW clause” on page 1051.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [\textit{SQL Anywhere Server - SQL Usage}].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [\textit{SQL Anywhere Server - SQL Usage}].

See also

- “CUME_DIST function [Ranking]” on page 200
- “DENSE_RANK function [Ranking]” on page 222
- “ROW_NUMBER function [Miscellaneous]” on page 353
- “PERCENT_RANK function [Ranking]” on page 316

Standards and compatibility

- SQL/2008 The RANK function is part of optional SQL/2008 language feature T612, "Advanced OLAP operations".

Example

The following example provides a rank in descending order of employees' salaries in Utah and New York. Notice that the 7th and 8th employees have an identical salary and therefore share the 7th place ranking. The employee that follows receives the 9th place ranking, which leaves a gap in the ranking sequence (no 8th place ranking).

\begin{verbatim}
SELECT Surname, Salary, State, 
RANK() OVER (ORDER BY Salary DESC) "Rank"
FROM GROUPO.Employees WHERE State IN ('NY','UT');
\end{verbatim}

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
Surname & Salary & State & Rank \\
\hline
Shishov & 72995.000 & UT & 1 \\
Wang & 68400.000 & UT & 2 \\
Cobb & 62000.000 & UT & 3 \\
\hline
\end{tabular}
\end{center}
<table>
<thead>
<tr>
<th>Surname</th>
<th>Salary</th>
<th>State</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morris</td>
<td>61300.000</td>
<td>UT</td>
<td>4</td>
</tr>
<tr>
<td>Davidson</td>
<td>57090.000</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Martel</td>
<td>55700.000</td>
<td>NY</td>
<td>6</td>
</tr>
<tr>
<td>Blaikie</td>
<td>54900.000</td>
<td>NY</td>
<td>7</td>
</tr>
<tr>
<td>Diaz</td>
<td>54900.000</td>
<td>NY</td>
<td>7</td>
</tr>
<tr>
<td>Driscoll</td>
<td>48023.690</td>
<td>UT</td>
<td>9</td>
</tr>
<tr>
<td>Hildebrand</td>
<td>45829.000</td>
<td>UT</td>
<td>10</td>
</tr>
<tr>
<td>Whitney</td>
<td>45700.000</td>
<td>NY</td>
<td>11</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Lynch</td>
<td>24903.000</td>
<td>UT</td>
<td>19</td>
</tr>
</tbody>
</table>

## READ_CLIENT_FILE function [String]

Reads data from the specified file on the client computer.

### Syntax

```
READ_CLIENT_FILE( client-filename-expression )
```

### Parameters

- **client-filename-expression**
  
  CHAR value indicating the name of the file on the client computer. The path is resolved on the client computer relative to the current working directory of the client application.

### Returns

LONG BINARY

### Remarks

The value returned by the READ_CLIENT_FILE function represents the contents of the specified client file. You can use the function in syntax wherever a BINARY expression is allowed.

Since the data returns as a binary string, if the data is in another character set, or is compressed, or is encrypted, you may also need to perform character set conversion, decompression, or decryption on it.

During evaluation of READ_CLIENT_FILE, the database server initiates the transfer of the specified file from the client. The client, upon receiving the transfer request, obtains a shared lock on the client file, and holds the lock until the database server requests the client to terminate the request.
Reading of the file is performed by the client software library, and the transfer of data is done using the command sequence communication protocol.

Privileges
When reading from a file on a client computer:

- You must have the READ FILE system privilege.
- Read permissions are required on the directory being read from.
- The allow_read_client_file database option must be enabled.
- The read_client_file secure feature must be enabled.

See also
- “-sf database server option” [SQL Anywhere Server - Database Administration]
- “allow_read_client_file option” [SQL Anywhere Server - Database Administration]
- “Access to data on client computers” [SQL Anywhere Server - SQL Usage]
- “DECOMPRESS function [String]” on page 219
- “DECRYPT function [String]” on page 220
- “CSCONVERT function [String]” on page 198

Standards and compatibility
- SQL/2008 Vendor extension.

REGEXP_SUBSTR function [String]
Extracts substrings from strings using regular expressions.

Syntax

REGEXP_SUBSTR( expression,
regular-expression

Parameters

- **expression** The string to be searched.
- **regular-expression** The pattern you are trying to match. For more information about regular expression syntax, see “Regular expressions overview” on page 26.
- **start-offset** The offset into expression at which to start searching. start-offset is expressed as a positive integer, and reflects the number of characters to count when starting from the left side of the string. The default is 1 (the start of the string).
- **occurrence-number** For multiple matches within expression, specify an integer indicating the occurrence to locate. For example, 3 finds the third occurrence. The default is 1.
• escape-expression  The escape character to use for regular-expression. The default is the backslash character (\).

Returns
LONG VARCHAR

Remarks
REGEXP_SUBSTR returns NULL if regular-expression is not found.

Similar to the REGEXP search condition, the REGEXP_SUBSTR function uses code points for matching and range evaluation. Database case sensitivity does not impact results. For more information about how REGEXP_SUBSTR performs matching and set evaluation, see “LIKE, REGEXP, and SIMILAR TO: Differences in character comparisons” on page 46.

When matching against a character class that contains only a sub-character class, include the outer square brackets and the square brackets for the sub-character class (for example, REGEXP_SUBSTR (expression, ['[:digit:]'])'). For more information about sub-character class matching, see “Regular expressions: Special sub-character classes” on page 30.

If start-offset is specified, that offset specifies the start of the expression to be matched. In particular, ^ matches the beginning of the expression starting at start-offset.

See also
• “Regular expressions syntax” on page 27
• “REGEXP search condition” on page 51

Standards and compatibility
• SQL/2008 Vendor extension. The corresponding function in the SQL/2008 standard is the SUBSTRING_REGEX function, which has similar parameters. SUBSTRING_REGEX is part of optional SQL/2008 language feature F844.

Example
The following statement breaks values in the Employees.Street column into street number and street name:

```
SELECT REGEXP_SUBSTR( Street, '^\S+' ) as street_num,
       REGEXP_SUBSTR( Street, '(?<=^\S+\s+).*$' ) AS street_name
FROM GROUPO.Employees;
```

<table>
<thead>
<tr>
<th>street_num</th>
<th>street_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>East Washington Street</td>
</tr>
<tr>
<td>7</td>
<td>Pleasant Street</td>
</tr>
<tr>
<td>539</td>
<td>Pond Street</td>
</tr>
<tr>
<td>1244</td>
<td>Great Plain Avenue</td>
</tr>
</tbody>
</table>
To determine whether the IP address of the current connection is in a range of IP addresses (in this case, 10.25.101.xxx or 10.25.102.xxx), you can execute the following statement:

```sql
IF REGEXP_SUBSTR( CONNECTION_PROPERTY( 'NodeAddress' ), '\d+\.\d+\.\d+' )
   IN ( '10.25.101', '10.25.102' ) THEN
   MESSAGE 'In range' TO CLIENT;
ELSE
   MESSAGE 'Out of range' TO CLIENT;
END IF;
```

**REGR_AVGX function [Aggregate]**

Computes the average of the independent variable of the regression line.

**Syntax 1**

```sql
REGR_AVGX( dependent-expression , independent-expression )
```

**Syntax 2**

```sql
REGR_AVGX( dependent-expression , independent-expression )
OVER ( window-spec )
```

*window-spec*: see Syntax 2 instructions in the Remarks section below

**Parameters**

- **dependent-expression**  The variable that is affected by the independent variable.
- **independent-expression**  The variable that influences the outcome.

**Returns**

DOUBLE

**Remarks**

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where \( x \) represents the independent-expression:

```sql
AVG( x )
```

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].
Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also

- “AVG function [Aggregate]” on page 164
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_INTERCEPT function [Aggregate]” on page 335
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_SLOPE function [Aggregate]” on page 338
- “REGR_SXX function [Aggregate]” on page 339
- “REGR_SXY function [Aggregate]” on page 341
- “REGR_SYY function [Aggregate]” on page 342
- “REGR_AVGY function [Aggregate]” on page 332
- “WINDOW clause” on page 1051

Standards and compatibility

- SQL/2008  REGR_AVGX is part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement calculates the average of the dependent variable, employee age:

```
SELECT REGR_AVGX( Salary, ( 2008 - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

REGR_AVGY function [Aggregate]

Computes the average of the dependent variable of the regression line.

Syntax 1

REGR_AVGY( dependent-expression , independent-expression )

Syntax 2

REGR_AVGY( dependent-expression , independent-expression )
OVER ( window-spec )

window-spec : see Syntax 2 instructions in the Remarks section below

Parameters

- dependent-expression  The variable that is affected by the independent variable.
• independent-expression  The variable that influences the outcome.

Returns
DOUBLE

Remarks
This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the dependent-expression:

\[
\text{AVG}(y)
\]

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also
• “REGR_COUNT function [Aggregate]” on page 334
• “REGR_INTERCEPT function [Aggregate]” on page 335
• “REGR_COUNT function [Aggregate]” on page 334
• “REGR_SLOPE function [Aggregate]” on page 338
• “REGR_SXX function [Aggregate]” on page 339
• “REGR_SXY function [Aggregate]” on page 341
• “REGR_SYY function [Aggregate]” on page 342
• “REGR_AVGX function [Aggregate]” on page 331
• “AVG function [Aggregate]” on page 164
• “WINDOW clause” on page 1051

Standards and compatibility
• SQL/2008  REGR_AVGY is part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example
The following statement calculates the average of the independent variable, employee salary:
SELECT REGR_AVGY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;

REGR_COUNT function [Aggregate]

Returns an integer that represents the number of non-NULL number pairs used to fit the regression line.

Syntax 1
REGR_COUNT( dependent-expression , independent-expression )

Syntax 2
REGR_COUNT( dependent-expression , independent-expression )
OVER ( window-spec )

window-spec : see Syntax 2 instructions in the Remarks section below

Parameters
- dependent-expression  The variable that is affected by the independent variable.
- independent-expression  The variable that influences the outcome.

Returns
INTEGER

Remarks
Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].
See also

- “WINDOW clause” on page 1051
- “REGR_INTERCEPT function [Aggregate]” on page 335
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_SLOPE function [Aggregate]” on page 338
- “REGR_SXX function [Aggregate]” on page 339
- “REGR_SXY function [Aggregate]” on page 341
- “REGR_SYY function [Aggregate]” on page 342
- “REGR_AVGY function [Aggregate]” on page 332
- “REGR_AVGX function [Aggregate]” on page 331
- “COUNT function [Aggregate]” on page 192
- “AVG function [Aggregate]” on page 164
- “SUM function [Aggregate]” on page 381

Standards and compatibility

- SQL/2008  
  REGR_COUNT is part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the number of non-NULL pairs that were used to fit the regression line:

```sql
SELECT REGR_COUNT( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

REGR_INTERCEPT function [Aggregate]

Computes the y-intercept of the linear regression line that best fits the dependent and independent variables.

Syntax 1

```sql
REGR_INTERCEPT( dependent-expression , independent-expression )
```

Syntax 2

```sql
REGR_INTERCEPT( dependent-expression , independent-expression )
OVER ( window-spec )
```

`window-spec`: see Syntax 2 instructions in the Remarks section below

Parameters

- `dependent-expression`  
  The variable that is affected by the independent variable.

- `independent-expression`  
  The variable that influences the outcome.

Returns

DOUBLE
Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where $y$ represents the dependent-expression and $x$ represents the independent-expression:

$$\text{AVG}(y) - \text{REGR}_\text{SLOPE}(y, x) \times \text{AVG}(x)$$

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also

- “WINDOW clause” on page 1051
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_SLOPE function [Aggregate]” on page 338
- “REGR_SXX function [Aggregate]” on page 339
- “REGR_SXY function [Aggregate]” on page 341
- “REGR_SYY function [Aggregate]” on page 342
- “REGR_AVGY function [Aggregate]” on page 332
- “REGR_AVGX function [Aggregate]” on page 331
- “REGR_SLOPE function [Aggregate]” on page 338
- “AVG function [Aggregate]” on page 164

Standards and compatibility

- SQL/2008
  REGR_INTERCEPT is part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the y-intercept of the linear regression line:

```sql
SELECT REGR_INTERCEPT( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```
REGR_R2 function [Aggregate]

Computes the coefficient of determination (also referred to as R-squared or the goodness of fit statistic) for the regression line.

Syntax 1

REGR_R2( dependent-expression , independent-expression )

Syntax 2

REGR_R2( dependent-expression , independent-expression )
OVER ( window-spec )

window-spec : see Syntax 2 instructions in the Remarks section below

Parameters

● dependent-expression  The variable that is affected by the independent variable.
● independent-expression  The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL.

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].
See also

- “WINDOW clause” on page 1051
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_INTERCEPT function [Aggregate]” on page 335
- “REGR_SLOPE function [Aggregate]” on page 338
- “REGR_SXX function [Aggregate]” on page 339
- “REGR_SXY function [Aggregate]” on page 341
- “REGR_SYY function [Aggregate]” on page 342
- “REGR_AVGX function [Aggregate]” on page 331
- “REGR_AVGY function [Aggregate]” on page 332

Standards and compatibility

- SQL/2008 REGR_R2 is part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the coefficient of determination for the regression line:

```
SELECT REGR_R2( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

**REGR_SLOPE function [Aggregate]**

Computes the slope of the linear regression line fitted to non-NULL pairs.

**Syntax 1**

```
REGR_SLOPE( dependent-expression , independent-expression )
```

**Syntax 2**

```
REGR_SLOPE( dependent-expression , independent-expression )
OVER ( window-spec )
```

*window-spec*: see Syntax 2 instructions in the Remarks section below

**Parameters**

- **dependent-expression** The variable that is affected by the independent variable.
- **independent-expression** The variable that influences the outcome.

**Returns**

DOUBLE

**Remarks**

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.
The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where \( y \) represents the dependent-expression and \( x \) represents the independent-expression:

\[
\text{COVAR\_POP}(y, x) / \text{VAR\_POP}(x)
\]

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spect can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also
- “WINDOW clause” on page 1051
- “REGR\_COUNT function [Aggregate]” on page 334
- “REGR\_INTERCEPT function [Aggregate]” on page 335
- “REGR\_COUNT function [Aggregate]” on page 334
- “REGR\_SXX function [Aggregate]” on page 339
- “REGR\_SXY function [Aggregate]” on page 341
- “REGR\_SYY function [Aggregate]” on page 342
- “REGR\_AVGX function [Aggregate]” on page 331
- “REGR\_AVGY function [Aggregate]” on page 332
- “COVAR\_POP function [Aggregate]” on page 195
- “VAR\_POP function [Aggregate]” on page 402

Standards and compatibility
- SQL/2008  REGR\_SLOPE is part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the value 935.3429749445614:

```sql
SELECT REGR\_SLOPE( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

**REGR\_SXX function [Aggregate]**

Returns the sum of squares of the independent expressions used in a linear regression model. The REGR\_SXX function can be used to evaluate the statistical validity of a regression model.
Syntax 1

```
REGR_SXX( dependent-expression , independent-expression )
```

Syntax 2

```
REGR_SXX( dependent-expression , independent-expression )
OVER ( window-spec )

window-spec : see Syntax 2 instructions in the Remarks section below
```

Parameters

- **dependent-expression**  The variable that is affected by the independent variable.
- **independent-expression**  The variable that influences the outcome.

Returns

DOUBLE

Remarks

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where \( y \) represents the dependent-expression and \( x \) represents the independent-expression:

\[
\text{REGR_COUNT}( y, x ) * \text{VAR_POP}( x )
\]

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].
See also
- “WINDOW clause” on page 1051
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_INTERCEPT function [Aggregate]” on page 335
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_AVGX function [Aggregate]” on page 331
- “REGR_AVGY function [Aggregate]” on page 332
- “REGR_SXY function [Aggregate]” on page 341
- “REGR_SYY function [Aggregate]” on page 342
- “VAR_POP function [Aggregate]” on page 402

Standards and compatibility
- **SQL/2008**  REGR_SXX is part of optional SQL/2008 language feature T621, "Enhanced numeric functions”.

Example
The following statement returns the value 5916.4800000000105:

```sql
SELECT REGR_SXX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

**REGR_SXY function [Aggregate]**

Returns the sum of products of the dependent and independent variables. The REGR_SXY function can be used to evaluate the statistical validity of a regression model.

**Syntax 1**

```sql
REGR_SXY( dependent-expression , independent-expression )
```

**Syntax 2**

```sql
REGR_SXY( dependent-expression , independent-expression )
OVER ( window-spec )
```

*window-spec*: see Syntax 2 instructions in the Remarks section below

**Parameters**
- **dependent-expression**  The variable that is affected by the independent variable.
- **independent-expression**  The variable that influences the outcome.

**Returns**

DOUBLE

**Remarks**

This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result. If the function is applied to an empty set, then it returns NULL.
The function is applied to the set of \((dependent-expression \text{ and } independent-expression)\) pairs after eliminating all pairs for which either \(dependent-expression\) or \(independent-expression\) is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where \(y\) represents the \(dependent-expression\) and \(x\) represents the \(independent-expression\):

\[
REGR_COUNT(y, x) * COVAR_POP(y, x)
\]

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of \(window-spec\) can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the \(window-spec\) definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also
- “WINDOW clause” on page 1051
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_INTERCEPT function [Aggregate]” on page 335
- “REGR_COUNT function [Aggregate]” on page 334
- “REGR_SLOPE function [Aggregate]” on page 338
- “REGR_AVGX function [Aggregate]” on page 331
- “REGR_AVGY function [Aggregate]” on page 332
- “REGR_SXX function [Aggregate]” on page 339
- “REGR_SYY function [Aggregate]” on page 342

Standards and compatibility

- **SQL/2008** RER_SXY is part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

Example

The following statement returns the sum of products of the dependent and independent variables:

```sql
SELECT REGR_SXY( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

**REGR_SYY function [Aggregate]**

Returns values that can evaluate the statistical validity of a regression model.

Syntax 1

```sql
REGR_SYY( dependent-expression, independent-expression )
```
Syntax 2
REGR_SYY( dependent-expression, independent-expression )
OVER ( window-spec )

window-spec : see Syntax 2 instructions in the Remarks section below

Parameters
● dependent-expression The variable that is affected by the independent variable.
● independent-expression The variable that influences the outcome.

Returns
DOUBLE

Remarks
This function converts its arguments to DOUBLE, and performs the computation in double-precision floating-point arithmetic. If the function is applied to an empty set, then it returns NULL.

The function is applied to the set of (dependent-expression and independent-expression) pairs after eliminating all pairs for which either dependent-expression or independent-expression is NULL. The function is computed simultaneously during a single pass through the data. After eliminating NULL values, the following computation is then made, where y represents the dependent-expression and x represents the independent-expression:

REGR_COUNT( y, x ) * VAR_POP( y )

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].
Standards and compatibility

- **SQL/2008**  
  REGR_SYY is part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

### Example

The following statement returns the value 26,708,672,843.3002:

```sql
SELECT REGR_SYY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM GROUPO.Employees;
```

### REMAINDER function [Numeric]

Returns the remainder when one whole number is divided by another.

#### Syntax

```
REMAINDER( dividend, divisor )
```

#### Parameters

- **dividend**  
  The dividend, or numerator of the division.

- **divisor**  
  The divisor, or denominator of the division.

#### Returns

- **INTEGER**
- **NUMERIC**

#### Remarks

You can also use the MOD function to return the remainder.

### See also

- “MOD function [Numeric]” on page 299

### Standards and compatibility

- **SQL/2008**  
  Vendor extension.
Example

The following statement returns the value 2:

    SELECT REMAINDER( 5, 3 );

**REPEAT function [String]**

Concatenates a string a specified number of times.

**Syntax**

    REPEAT( string-expression, integer-expression )

**Parameters**

- **string-expression**  The string to be repeated.
- **integer-expression**  The number of times the string is to be repeated. If `integer-expression` is negative, an empty string is returned.

**Returns**

- LONG VARCHAR
- LONG NVARCHAR

**Remarks**

If the actual length of the result string exceeds the maximum for the return type, an error occurs. The result is truncated to the maximum string size allowed.

The behavior of this function is identical to that of the REPLICATE function.

This function supports NCHAR inputs and/or outputs.

**See also**

- “REPLICATE function [String]” on page 347
- “String functions” on page 152

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

The following statement returns the value `repeat`\_`repeat`\_`repeat`:

    SELECT REPEAT( 'repeat', 3 );

**REPLACE function [String]**

Replaces a string with another string, and returns the new results.
Syntax

REPLACE( original-string, search-string, replace-string )

Parameters

If any argument is NULL, the function returns NULL.

- **original-string**  The string to be searched. This can be any length.
- **search-string**  The string to be searched for and replaced with replace-string. This string is limited to 255 bytes. If search-string is an empty string, the original string is returned unchanged.
- **replace-string**  The replacement string, which replaces search-string. This can be any length. If replacement-string is an empty string, all occurrences of search-string are deleted.

Returns

- LONG VARCHAR
- LONG NVARCHAR

Remarks

This function replaces all occurrences.

Comparisons are case-sensitive on case-sensitive databases.

This function supports NCHAR inputs and/or outputs.

See also

- “SUBSTRING function [String]” on page 379
- “CHARINDEX function [String]” on page 177
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the value xx.def.xx.ghi:

```sql
SELECT REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' );
```

The following statement generates a result set containing ALTER PROCEDURE statements which, when executed, would repair stored procedures that reference a table that has been renamed. (To be useful, the table name must be unique.)

```sql
SELECT REPLACE(
    REPLACE( proc_defn, 'OldTableName', 'NewTableName' ),
    'CREATE PROCEDURE',
    'ALTER PROCEDURE')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%';
```
REPLICATE function [String]
Concatenates a string a specified number of times.

Syntax
REPLICATE( string-expression, integer-expression )

Parameters
- string-expression  The string to be repeated.
- integer-expression  The number of times the string is to be repeated.

Returns
- LONG VARCHAR
- LONG NVARCHAR

Remarks
If the actual length of the result string exceeds the maximum for the return type, an error occurs. The result is truncated to the maximum string size allowed.

The behavior of this function is identical to that of the REPEAT function.

This function supports NCHAR inputs and/or outputs.

See also
- “REPEAT function [String]” on page 345
- “String functions” on page 152

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement returns the value repeatrepeatrepeat:

SELECT REPLICATE( 'repeat', 3 );

REVERSE function [String]
Returns the reverse of a character expression.

Syntax
REVERSE( string-expression )

Parameters
- string-expression  The string to be reversed.
Returns

- LONG VARCHAR
- LONG NVARCHAR

Remarks

This function supports NCHAR inputs and/or outputs.

See also

- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the value cba:

```
SELECT REVERSE( 'abc' );
```

**REWRITE function [Miscellaneous]**

Returns a rewritten SELECT, UPDATE, or DELETE statement.

Syntax

```
REWRITE( select-statement [, 'ANSI' ] )
```

Parameters

- **select-statement**  The SQL statement to which the rewrite optimizations are applied to generate the function's results.

Returns

LONG VARCHAR

Remarks

You can use the REWRITE function without the ANSI argument to help understand how the optimizer generated the access plan for a given query. In particular, you can find how SQL Anywhere has rewritten the conditions in the statement's WHERE, ON, and HAVING clauses, and then determine if applicable indexes exist that can be exploited to improve the request execution time.

The statement that is returned by REWRITE may not match the semantics of the original statement. This is because several rewrite optimizations introduce internal mechanisms that cannot be translated directly into SQL. For example, the server's use of row identifiers to perform duplicate elimination cannot be translated into SQL.

The rewritten query from the REWRITE function is not intended to be executable. It is a tool for analyzing performance issues by showing what gets passed to the optimizer after the rewrite phase.
There are some rewrite optimizations that are not reflected in the output of REWRITE. They include LIKE optimization, optimization for minimum or maximum functions, upper/lower elimination, and predicate subsumption.

If ANSI is specified, REWRITE returns the ANSI equivalent to the statement. In this case, only the following rewrite optimizations are applied:

- Transact-SQL outer joins are rewritten as ANSI SQL outer joins.
- Duplicate correlation names are eliminated.
- KEY and NATURAL joins are rewritten as ANSI SQL joins.

See also
- "Optimizations performed during query processing" [SQL Anywhere Server - SQL Usage]
- "extended_join_syntax option" [SQL Anywhere Server - Database Administration]
- “Transact-SQL outer joins (*= or =*)” [SQL Anywhere Server - SQL Usage]
- “Key joins” [SQL Anywhere Server - SQL Usage]
- “Natural joins” [SQL Anywhere Server - SQL Usage]
- “Duplicate correlation names in joins (star joins)” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
In the following statement, two rewrite optimizations are performed on a query. The first is the un-nesting of the subquery into a join between the Employees and SalesOrders tables. The second optimization simplifies the query by eliminating the primary key - foreign key join between Employees and SalesOrders. Part of this rewrite optimization is to replace the join predicate e.EmployeeID=s.SalesRepresentative with the predicate s.SalesRepresentative IS NOT NULL.

```sql
SELECT REWRITE( 'SELECT s.ID, s.OrderDate
FROM GROUPO.SalesOrders s
WHERE EXISTS ( SELECT *
  FROM GROUPO.Employees e
  WHERE e.EmployeeID = s.SalesRepresentative) ' ) FROM dummy;
```

The query returns a single column result set containing the rewritten query:

```
'SELECT s.ID, s.OrderDate FROM GROUPO.SalesOrders s WHERE s.SalesRepresentative IS NOT NULL'
```

The next REWRITE statement uses the ANSI argument:

```sql
SELECT REWRITE( 'SELECT DISTINCT s.ID, s.OrderDate, e.GivenName, e.EmployeeID
FROM GROUPO.SalesOrders s, GROUPO.Employees e
WHERE e.EmployeeID *= s.SalesRepresentative', 'ANSI' ) FROM dummy;
```

The result is the ANSI equivalent of the statement. In this case, the Transact-SQL outer join is converted to an ANSI outer join. The query returns a single column result set (broken into separate lines for readability):
'SELECT DISTINCT s.ID, s.OrderDate, e.GivenName, e.EmployeeID
FROM GROUPO.Employees as e
LEFT OUTER JOIN GROUPO.SalesOrders as s
ON e.EmployeeID = s.SalesRepresentative;
ROUND function [Numeric]

Rounds the numeric-expression to the specified integer-expression amount of places after the decimal point.

Syntax

```
ROUND( numeric-expression, integer-expression )
```

Parameters

- **numeric-expression**  The number, passed into the function, to be rounded.
- **integer-expression**  A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative expression specifies the number of significant digits to the left of the decimal point at which to round.

Returns

NUMERIC

Remarks

The result of this function is either numeric or double. When there is a numeric result and the integer integer-expression is a negative value, the precision is increased by one.

See also

- “TRUNCNUM function [Numeric]” on page 394

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the value 123.200:

```
SELECT ROUND( 123.234, 1 );
```

ROW constructor [Composite]

Returns a sequence of (field name data type, ... ) pairs named fields.

Syntax 1

```
ROW( expression [, expression ... ] )
```

Syntax 2

```
ROW( single-row-query-expression )
```

Parameters

- **expression**  An expression representing a single field.
• **single-row-query-expression**  A query statement that returns a single field.

**Returns**

Field value

**Remarks**

All fields are initialized to NULL, and remain NULL until a value is explicitly or implicitly placed within a particular field.

ROW types cannot be specified in the outermost SELECT list of a view definition, or in a top-level SELECT block or query expression that is returned to the client. ROW types cannot be stored as columns in a base or temporary table.

A row type can contain arbitrary levels of nesting, resulting in complex structured types. Each row sub-type is given a name, and these names can be referenced by using dot-notation to reference particular values. For example:

```
DECLARE sample ROW( x INT, w ROW( y INT, z INT ) );
SET Sample = ROW( 3, ROW( 6,7 ) );
SELECT (Sample).w.y FROM dummy;
```

Dot-notation permits other expressions in the same or nested query block to refer to all or portions of a row type for comparison or initialization of other ROW types.

When referring to fields within columns with a qualified name, surround column names with parentheses. In the following example, the myrowcolumn column is qualified by the name of its derived table:

```
SELECT ( myderivedtable.myrowcolumn ).id FROM
(SELECT ROW( id, name )
 FROM GROUPO.Product )
AS myderivedtable( myrowcolumn );
```

**See also**

- “Composite data types” on page 128
- “Comparisons of composite types” on page 136
- “CAST function [Data type conversion]” on page 174

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

**Example**

The following statement illustrates how to construct a ROW type that contains structure types of product information for each product in a Products table.

```
SELECT ROW( ID, NAME, DESCRIPTION ) AS pInfo FROM GROUPO.Products;
```

The pInfo row type has elements named ID, NAME, and DESCRIPTION, which are borrowed from the attribute names from the Products table.

The following statement illustrates how to use the CAST function to assign explicit names to an implicit ROW type:
SELECT CAST( ROW( 9, 'Tee Shirt', 'My tee shirt' )
    AS ROW(ProductID INTEGER,
            ProductName CHAR(25),
            ProductDescription CHAR(35)
    )
) AS pInfo FROM dummy;

SQL expressions that use this query result as input can reference the components of the pInfo row by
dotted expressions on their names. ROW types can also be constructed with a single-row query
expression.

The following statement illustrates an alternative way of constructing a ROW:

   SELECT ROW( SELECT Name, Quantity FROM Products WHERE ID = 300 );

The following statement illustrates how to set a field, last_name, in a ROW, student, by name:

   SET student.last_name = 'Johnson';

**ROW_NUMBER function [Miscellaneous]**

Assigns a unique number to each row. Use this function instead of the NUMBER function.

**Syntax**

```
ROW_NUMBER( ) OVER ( window-spec )
```

*window-spec*: see the Remarks section below

**Returns**

INTEGER

**Remarks**

Elements of *window-spec* can be specified either in the function syntax (inline), or with a WINDOW
clause in the SELECT statement. When used as a window function, you must specify an ORDER BY
clause, you may specify a PARTITION BY clause, however, you cannot specify a ROWS or RANGE
clause. See the *window-spec* definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples,
see “Window functions” [*SQL Anywhere Server - SQL Usage*].

For more information about specifying a window specification in an OVER clause, see “Window
definition: Inlining using the OVER clause and WINDOW clause” [*SQL Anywhere Server - SQL Usage*].

**See also**

- “WINDOW clause” on page 1051
- “NUMBER function [Miscellaneous]” on page 313
- “RANK function [Ranking]” on page 326
- “ROWID function [Miscellaneous]” on page 354
Standards and compatibility

- **SQL/2008**  ROW_NUMBER is part of optional SQL/2008 language feature T611, "Elementary OLAP operations".

Example

The following statement returns a result set that provides unique row numbers for each employee in New York and Utah. Because the query is ordered by Salary in descending order, the first row number is given to the employee with the highest salary in the data set. Although two employees have identical salaries, the tie is not resolved because the two employees are assigned unique row numbers.

```sql
SELECT Surname, Salary, State,
ROW_NUMBER() OVER (ORDER BY Salary DESC) "Rank"
FROM GROUPO.Employees WHERE State IN ('NY','UT');
```

<table>
<thead>
<tr>
<th>Surname</th>
<th>Salary</th>
<th>State</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shishov</td>
<td>72995.000</td>
<td>UT</td>
<td>1</td>
</tr>
<tr>
<td>Wang</td>
<td>68400.000</td>
<td>UT</td>
<td>2</td>
</tr>
<tr>
<td>Cobb</td>
<td>62000.000</td>
<td>UT</td>
<td>3</td>
</tr>
<tr>
<td>Morris</td>
<td>61300.000</td>
<td>UT</td>
<td>4</td>
</tr>
<tr>
<td>Davidson</td>
<td>57090.000</td>
<td>NY</td>
<td>5</td>
</tr>
<tr>
<td>Martel</td>
<td>55700.000</td>
<td>NY</td>
<td>6</td>
</tr>
<tr>
<td>Blaikie</td>
<td>54900.000</td>
<td>NY</td>
<td>7</td>
</tr>
<tr>
<td>Diaz</td>
<td>54900.000</td>
<td>NY</td>
<td>8</td>
</tr>
<tr>
<td>Driscoll</td>
<td>48023.690</td>
<td>UT</td>
<td>9</td>
</tr>
<tr>
<td>Hildebrand</td>
<td>45829.000</td>
<td>UT</td>
<td>10</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Lynch</td>
<td>24903.000</td>
<td>UT</td>
<td>19</td>
</tr>
</tbody>
</table>

**ROWID function [Miscellaneous]**

Returns an UNSIGNED BIGINT value that uniquely identifies a row within a table.

**Syntax**

```sql
ROWID( correlation-name )
```
Parameters

- **correlation-name**  The correlation name of a table used in the query. The correlation name should refer to a base table, a temporary table, a global temporary table or a proxy table (permitted only when the underlying proxy server supports a similar function). The argument of the ROWID function should not refer to a view, derived table, common table expression or a procedure.

Returns

UNSIGNED BIGINT

Remarks

Returns the row identifier of the row in the table corresponding to the given correlation name.

The value returned by the function is not necessarily constant between queries as various operations performed on the database may result in changes to the row identifiers of a table. In particular, the REORGANIZE TABLE statement is likely to result in changes to row identifiers. Additionally, row identifiers may be reused after a row has been deleted. So, users should refrain from using the ROWID function in ordinary situations; retrieval by primary key value should be used instead. It is recommended that ROWID be used only in diagnostic situations.

Although the result of this function is an UNSIGNED BIGINT, the results of most arithmetic operations on this value have no particular meaning. For example, you should not expect that adding one to a row identifier will give you the row identifier of the next row. Also, only equality and IN predicates are sargable if they involve the use of ROWID. If necessary, predicates involving ROWID, such as ROWID( T ) = literal, may be used to cast to a 64-bit UNSIGNED INTEGER value. If the conversion cannot be performed a data exception will occur. If the value of literal is an invalid row identifier then the comparison predicate evaluates to FALSE.

The ROWID function cannot be used inside a CHECK constraint on either a table or a column, nor can it be used in the COMPUTE expression for a computed column.

See also

- “ROW_NUMBER function [Miscellaneous]” on page 353

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the row identifier of the row in Employee where id is equal to 105:

```sql
SELECT ROWID( Employees ) FROM GROUPO.Employees WHERE Employees.EmployeeID = 105;
```

The following statement returns a list of the locks on rows in the Employees table along with the contents of those rows:

```sql
SELECT *
FROM sa_locks() S JOIN GROUPO.Employees WITH( NOLOCK )
ON ROWID( Employees ) = S.row_identifier
WHERE S.table_name = 'Employees';
```
RTRIM function [String]

Removes trailing blanks from the string.

Syntax

RTRIM( string-expression )

Parameters

● string-expression  The string to be trimmed.

Returns

● VARCHAR
● NVARCHAR
● LONG VARCHAR
● LONG NVARCHAR

Remarks

The actual length of the result is the length of the expression minus the number of characters removed. If all the characters are removed, the result is an empty string.

If the argument is null, the result is the NULL value.

This function supports NCHAR inputs and/or outputs.

See also

● “TRIM function [String]” on page 393
● “LTRIM function [String]” on page 291
● “String functions” on page 152

Standards and compatibility

● SQL/2008  Vendor extension.

The TRIM specifications defined by the SQL/2008 standard (LEADING and TRAILING) are supplied by the SQL Anywhere LTRIM and RTRIM functions respectively.

Example

The following statement returns the string Test Message, with all trailing blanks removed:

SELECT RTRIM( 'Test Message     ' );

SECOND function [Date and time]

Returns the seconds value of the TIMESTAMP argument.

Syntax

SECOND( timestamp-expression )
Parameters

- **timestamp-expression**  The TIMESTAMP value.

Returns

SMALLINT

Remarks

Returns a number from 0 to 59 corresponding to the seconds component of the given TIMESTAMP argument value.

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statement returns the value 25.

```sql
```

**SECONDS function [Date and time]**

Manipulates a TIMESTAMP or returns the number of seconds between two TIMESTAMP values. See the Remarks section below.

Syntax 1

```sql
SECONDS( timestamp-expression )
```

Syntax 2

```sql
SECONDS( timestamp-expression, timestamp-expression )
```

Syntax 3

```sql
SECONDS( time-or-timestamp-expression, integer-expression )
```

Parameters

- **timestamp-expression**  A TIMESTAMP value.

- **time-or-timestamp-expression**  A value of type TIME or TIMESTAMP.

- **integer-expression**  The number of seconds to be added to the time-or-timestamp-expression. If integer-expression is negative, the appropriate number of seconds is subtracted from time-or-timestamp-expression. If you supply an integer expression, the time-or-timestamp-expression must be explicitly cast as a TIME, DATE, or TIMESTAMP data type. If time-or-timestamp-expression is a DATE type, its time portion is assumed to be midnight.

Returns

UNSIGNED BIGINT with Syntax 1.
SIGNED BIGINT with Syntax 2.

TIME or TIMESTAMP with Syntax 3.

Remarks
The result of the SECONDS function depends on its arguments.

- **Syntax 1** If you pass a single `timestamp-expression` to the SECONDS function, it will return the number of seconds between midnight 0000-02-29 and `timestamp-expression` as an UNSIGNED BIGINT.

  **Note**
  0000-02 is not meant to imply an actual date; it is the default date used by the SECONDS function.

- **Syntax 2** If you pass two TIMESTAMP values to the SECONDS function, the function returns the integer number of seconds between them as a SIGNED BIGINT value.

- **Syntax 3** If you pass a TIMESTAMP value and a INTEGER value to the SECONDS function, the function returns the TIMESTAMP result of adding the integer number of seconds to `time-or-timestamp-expression`. Similarly, if you pass a TIME value to the SECONDS function, the function returns a value of type TIME.

Instead of Syntax 2, use the DATEDIFF function. Instead of Syntax 3, use the DATEADD function.

See also
- “CAST function [Data type conversion]” on page 174
- “DATEADD function [Date and time]” on page 204
- “DATEDIFF function [Date and time]” on page 205

Standards and compatibility
- **SQL/2008** Vendor extension.

Example
The following statements return the value 14400, signifying that the second TIMESTAMP value is 14400 seconds after the first.

```sql
SELECT SECONDS( '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
SELECT DATEDIFF( second, '1999-07-13 06:07:12', '1999-07-13 10:07:12' );
```

The following statement returns the value 63062431632.

```sql
SELECT SECONDS( '1998-07-13 06:07:12' );
```

The following statements return the TIMESTAMP value 1999-05-12 21:05:12.000.

```sql
SELECT SECONDS( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5 );
SELECT DATEADD( second, 5, '1999-05-12 21:05:07' );
```
**SET_BIT function [Bit array]**

Sets the value of a specific bit in a bit array.

**Syntax**

```sql
SET_BIT( [ bit-expression, ]bit-position [, value ] )
```

**Parameters**

- **bit-expression**  The bit array in which to change the bit.
- **bit-position**  The position of the bit to be set. This must be an unsigned integer.
- **value**  The value to which the bit is to be set.

**Returns**

LONG VARBIT

**Remarks**

The default value of `bit-expression` is a bit array of length `bit-position`, containing all bits set to 0 (FALSE).

The default value of `value` is 1 (TRUE).

The result is NULL if any parameter is NULL.

The positions in the array are counted from the left side, starting at 1.

**See also**

- “GET_BIT function [Bit array]” on page 249
- “SET_BITS function [Aggregate]” on page 360
- “INTEGER data type” on page 102
- “Bitwise operators” on page 19
- “sa_get_bits system procedure” on page 1141

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

The following statement returns the value 00100011:

```sql
SELECT SET_BIT( '00110011', 4 , 0);
```

The following statement returns the value 00111011:

```sql
SELECT SET_BIT( '00110011', 5 , 1);
```

The following statement returns the value 00111011:

```sql
SELECT SET_BIT( '00110011', 5 );
```
The following statement returns the value 00001:

```
SELECT SET_BIT( 5 );
```

**SET_BITS function [Aggregate]**

Creates a bit array where specific bits, corresponding to values from a set of rows, are set to 1 (TRUE).

**Syntax**

```
SET_BITS( expression )
```

**Parameters**

- `expression` The expression used to determine which bits to set to 1. This is typically a column name.

**Returns**

LONG VARBIT

**Remarks**

Rows where the specified values are NULL are ignored.

If there are no rows, NULL is returned.

The length of the result is the largest position that was set to 1.

The `SET_BITS` function is equivalent to, but faster than, the following statement:

```
SELECT BIT_OR( SET_BIT( expression ) )
FROM table;
```

**See also**

- “Bitwise operators” on page 19
- “GET_BIT function [Bit array]” on page 249
- “SET_BIT function [Bit array]” on page 359
- “sa_get_bits system procedure” on page 1141

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statements return a bit array with the 2nd, 5th, and 10th bits set to 1 (or 0100100001):

```
CREATE TABLE t( r INTEGER );
INSERT INTO t values( 2 );
INSERT INTO t values( 5 );
INSERT INTO t values(10 );
SELECT SET_BITS( r ) FROM t;
```
SIGN function [Numeric]

Returns the sign (positive or negative) of the given number.

Syntax

```
SIGN( numeric-expression )
```

Parameters

- `numeric-expression` The number for which the sign is to be returned. `numeric-expression` may be of type INTEGER, DOUBLE, or NUMERIC.

Returns

SMALLINT

Remarks

- For negative numbers, the SIGN function returns -1.
- For zero, the SIGN function returns 0.
- For positive numbers, the SIGN function returns 1.

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns the value -1

```
SELECT SIGN( -550 );
```

SIMILAR function [String]

Returns a number indicating the similarity between two strings.

Syntax

```
SIMILAR( string-expression-1, string-expression-2 )
```

Parameters

- `string-expression-1` The first string to be compared.
- `string-expression-2` The second string to be compared.

Returns

SMALLINT
Remarks

The function returns an integer between 0 and 100 representing the similarity between the two strings. The result can be interpreted as the percentage of characters matched between the two strings. A value of 100 indicates that the two strings are identical.

This function can be used to correct a list of names (such as customers). Some customers may have been added to the list more than once with slightly different names. You can use the SIMILAR function to find similar customer names by joining the customer table to itself, producing a report of all similarities greater than 90 percent, but less than 100 percent.

The calculation performed for the SIMILAR function is more complex than just the number of characters that match.

See also

- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement returns the value 75, indicating that the two values are 75% similar.

```sql
SELECT SIMILAR( 'toast', 'coast' );
```

SIN function [Numeric]

Returns the sine of a number.

Syntax

```
SIN( numeric-expression )
```

Parameters

- `numeric-expression`  The angle, in radians.

Returns

DOUBLE

Remarks

The SIN function returns the sine of the argument, where the argument is an angle expressed in radians. The SIN and ASIN functions are inverse operations.

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.
See also
● “ASIN function [Numeric]” on page 161
● “COS function [Numeric]” on page 191
● “COT function [Numeric]” on page 191
● “TAN function [Numeric]” on page 385

Standards and compatibility
● SQL/2008 Vendor extension.

Example
The following statement returns the SIN value of 0.52.

```
SELECT SIN( 0.52 );
```

**SOAP_HEADER function [SOAP]**

Returns a SOAP header entry, or an attribute value for a header entry of the SOAP request.

**Syntax**

```
SOAP_HEADER( header-key [, index, header-attribute ] )
```

**Parameters**
- **header-key** This VARCHAR parameter specifies the XML local name of the top level XML element for a given SOAP header entry.
- **index** This optional INTEGER parameter differentiates between SOAP header fields that have identical names. This can occur when multiple header entries have top level XML elements with the same localname. Usually, such elements have unique namespaces.
- **header-attribute** This optional VARCHAR parameter can specify any attribute node within a header entry element, including:
  - @namespace A special SQL Anywhere attribute used to access the namespace of the given header entry.
  - mustUnderstand A SOAP 1.1 header entry attribute indicating whether a header entry is mandatory or optional for the recipient to process.
  - encodingStyle A SOAP 1.1 header entry attribute indicating the encoding style. This attribute may be accessed, but it is not used internally by SQL Anywhere.
  - actor A SOAP 1.1 header entry attribute indicating the intended recipient of a header entry by specifying the recipient's URL.

**Returns**

LONG VARCHAR
Remarks

This function may be used with a single parameter header-key to return a header entry. A header entry is an XML string representation of an element, and all its sub-elements, contained within a SOAP header.

This function may also be used to extract a header entry attribute by specifying the optional index and header-attribute parameters.

This function returns the value of the named SOAP header field, or NULL if not called from a SOAP service. It is used when processing a SOAP request via a web service.

If a header for the given header-key does not exist, the return value is NULL.

See also

- “NEXT_SOAP_HEADER function [SOAP]” on page 310
- “sa_set_soap_header system procedure” on page 1244
- “Tutorial: Using SQL Anywhere to access a SOAP/DISH service” [SQL Anywhere Server - Programming]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Standards and compatibility

- SQL/2008 Vendor extension.

Example

When used within a stored procedure that is called by an HTTP web service, the following example processes all the keys located in the SOAP request header. When it processes the Authentication key, it also obtains the key's value.

```sql
BEGIN
    DECLARE hd_key LONG VARCHAR;
    DECLARE hd_entry LONG VARCHAR;
    header_loop:
        LOOP
            SET hd_key = NEXT_SOAP_HEADER( hd_key );
            IF hd_key IS NULL THEN
                -- no more header entries
                LEAVE header_loop;
            END IF;
            IF hd_key = 'Authentication' THEN
                SET hd_entry = SOAP_HEADER( hd_key );
            END IF;
        END LOOP header_loop;
END;
```

SORTKEY function [String]

Generates sort key values. That is, values that can be used to sort character strings based on alternate collation rules.
Syntax

\begin{verbatim}
SORTKEY( string-expression
  [, { collation-id
      | collation-name[ ( collation-tailoring-string) ] } ] )
\end{verbatim}

Parameters

- **string-expression**  The string expression must contain characters that are encoded in the database's character set.

  If *string-expression* is an empty string, the SORTKEY function returns a zero-length binary value. If *string-expression* is NULL, the SORTKEY function returns a NULL value. An empty string has a different sort order value than a NULL string from a database column.

  The maximum length of the string that the SORTKEY function can handle is 254 bytes. Any longer part is ignored.

- **collation-name**  A string or a character variable that specifies the name of the sort order to use. You can also specify the alias char\_collation, or, equivalently, db\_collation, to generate sortkeys as used by the CHAR collation in use by the database. Similarly, you can specify the alias nchar\_collation to generate sortkeys as used by the NCHAR collation in use by the database.

- **collation-id**  A variable, integer constant, or string that specifies the ID number of the sort order to use. If you do not specify *collation-id*, the default is Default Unicode multilingual.

- **collation-tailoring-string**  Optionally, you can specify collation tailoring options (*collation-tailoring-string*) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs assembled in parentheses, following the collation name. For example, `'UCA(locale=es;case=LowerFirst;accent=respect)'`. The syntax for specifying these options is identical to the syntax defined for the COLLATION clause of the CREATE DATABASE statement.

  **Note**
  All the collation tailoring options are supported when specifying the UCA collation. For all other collations, only case sensitivity tailoring is supported.

Returns

BINARY

Remarks

The SORTKEY function generates values that can be used to order results based on predefined sort order behavior. This allows you to work with character sort order behaviors that may not be available from the database collation. The returned value is a binary value that contains coded sort order information for the input string that is retained from the SORTKEY function. For example, you can store the values returned by the SORTKEY function in a column with the source character string. When you want to retrieve the character data in the desired order, the SELECT statement only needs to include an ORDER BY clause on the columns that contain the results of running the SORTKEY function.
The SORTKEY function guarantees that the values it returns for a given set of sort order criteria work for the binary comparisons that are performed on VARBINARY data types.

Generating sortkeys for queries can be expensive. As an alternative for frequently requested sortkeys, consider creating a computed column to hold the sortkey values, and then referencing that column in the ORDER BY clause of the query.

The input of the SORTKEY function can generate up to six bytes of sort order information for each input character. The output of the SORTKEY function is of type VARBINARY and has a maximum length of 1024 bytes.

When specifying UCA for the collation during sort key generation, by default, collation tailorings are accent and case sensitive. For example, when UCA is specified by itself, the default tailoring applied is equivalent to 'UCA(case=UpperFirst;accent=Respect;punct=Primary)'.

If a different tailoring is provided in the second parameter to SORTKEY, those settings override the default settings. For example, the following two statements are equivalent:

```sql
SELECT SORTKEY( 'abc', 'UCA(accent=Ignore)' );
SELECT SORTKEY( 'abc', 'UCA(case=UpperFirst;accent=Ignore;punct=Primary)' );
```

When specifying a non-UCA collation, by default, collation tailorings are also accent and case sensitive. However, for non-UCA collations, only the case sensitivity can be overridden using a collation tailoring. For example:

```sql
SELECT SORTKEY( 'abc', '1252LATIN1(case=Respect)' );
```

If the database was created without specifying tailoring options (for example, `dbinit -c -zn uca -dba DBA, sql mydb.db`), the following two clauses may generate different sort orders, even if the database collation name is specified for the SORTKEY function:

```sql
ORDER BY string-expression
ORDER BY SORTKEY( string-expression, database-collation-name )
```

This is because the default tailoring settings used for database creation and for the SORTKEY function are different. To get the same behavior from SORTKEY as for the database collation, either provide a tailoring syntax for `collation-tailoring-string` that matches the settings for the database collation, or specify `db_collation` for `collation-name`. For example:

```sql
SORTKEY( expression, 'db_collation' )
```

**Note**
Sort key values are generated differently depending on the version of SQL Anywhere. This can cause sorting issues if sort key values created by one version of SQL Anywhere are used in a database created by a different version of SQL Anywhere. You should regenerate sort key values if sorting issues occur.

You should also regenerate sort key values when upgrading your database using unload/reload.
See also

- “sort_collation option” [SQL Anywhere Server - Database Administration]
- “Alternate collations” [SQL Anywhere Server - Database Administration]
- “Collation tailoring options” [SQL Anywhere Server - Database Administration]
- “Recommended character sets and collations” [SQL Anywhere Server - Database Administration]
- “COMPARE function [String]” on page 179
- “CREATE DATABASE statement” on page 549
- “International languages and character sets” [SQL Anywhere Server - Database Administration]
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statements queries the Employees table and returns the FirstName and Surname of all employees, sorted by the sortkey values for the Surname column using the dict collation (Latin-1, English, French, German dictionary).

```
SELECT Surname, GivenName FROM GROUPO.Employees ORDER BY SORTKEY( Surname, 'dict' );
```

The following example returns the sortkey value for abc, using the UCA collation and tailoring options.

```
SELECT SORTKEY( 'abc', 'UCA(locale=es;case=LowerFirst;accent=respect)' );
```

SOUNDEX function [String]

Returns a number representing the sound of a string.

**Syntax**

```
SOUNDEX( string-expression )
```

**Parameters**

- **string-expression**  The string to be evaluated.

**Returns**

SMALLINT

**Remarks**

The SOUNDEX function value for a string is based on the first letter and the next three consonants other than H, Y, and W. Vowels in *string-expression* are ignored unless they are the first letter of the string. Doubled letters are counted as one letter. For example, the word "apples" is based on the letters A, P, L, and S.

Multibyte characters are ignored by the SOUNDEX function.

Although it is not perfect, the SOUNDEX function normally returns the same number for words that sound similar and that start with the same letter.
The SOUNDEX function works best with English words. It is less useful for other languages.

See also
- “String functions” on page 152

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement returns two identical numbers, 3827, representing the sound of each name.

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' );
```

**SPACE function [String]**

Returns a specified number of spaces.

Syntax
```
SPACE( integer-expression )
```

Parameters
- **integer-expression**  The number of spaces to return.

Returns
LONG VARCHAR

Remarks
If integer-expression is negative, a null string is returned.

See also
- “String functions” on page 152

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement returns a string containing 10 spaces.

```
SELECT SPACE( 10 );
```

**SQLDIALECT function [Miscellaneous]**

Returns either Watcom SQL or Transact-SQL, to indicate the SQL dialect of a statement.
Syntax

\texttt{SQLDIALECT( sql-statement-string )}

Parameters

\begin{itemize}
  \item \texttt{sql-statement-string} \quad The SQL statement that the function uses to determine its dialect.
\end{itemize}

Returns

\texttt{LONG VARCHAR}

See also

\begin{itemize}
  \item “TRANSACTSQL function [Miscellaneous]” on page 391
  \item “WATCOMSQL function [Miscellaneous]” on page 406
\end{itemize}

Standards and compatibility

\begin{itemize}
  \item SQL/2008 \quad Vendor extension.
\end{itemize}

Example

The following statement returns the string Transact-SQL.

\begin{verbatim}
SELECT
    SQLDIALECT( 'SELECT EmployeeName = Surname FROM GROUPO.Employees' )
FROM dummy;
\end{verbatim}

\textbf{SQLFLAGGER function [Miscellaneous]}

Returns the conformity of a given SQL statement to a specified standard.

Syntax

\texttt{SQLFLAGGER( sql-standard-string, sql-statement-string )}

Parameters

\begin{itemize}
  \item \texttt{sql-standard-string} \quad The standard level against which to test compliance. Possible values are the same as for the sqlflagger_error_level database option:
    \begin{itemize}
      \item \texttt{SQL:2008/Core} \quad Test for conformance to core SQL/2008 syntax.
      \item \texttt{SQL:2008/Package} \quad Test for conformance to full SQL/2008 syntax.
      \item \texttt{SQL:2003/Core} \quad Test for conformance to core SQL/2003 syntax.
      \item \texttt{SQL:2003/Package} \quad Test for conformance to full SQL/2003 syntax.
      \item \texttt{SQL:1999/Core} \quad Test for conformance to core SQL/1999 syntax.
      \item \texttt{SQL:1999/Package} \quad Test for conformance to full SQL/1999 syntax.
      \item \texttt{SQL:1992/Entry} \quad Test for conformance to entry-level SQL/1992 syntax.
    \end{itemize}
\end{itemize}
SQL functions

- **SQL:1992/Full**  Test for conformance to full-SQL/1992 syntax.
- **UltraLite**  Test for conformance to UltraLite.

- **sql-statement-string**  The SQL statement to check for conformance.

**Returns**

LONG VARCHAR

**See also**

- “sql_flagger_error_level option” [SQL Anywhere Server - Database Administration]
- “The SQL preprocessor” [SQL Anywhere Server - Programming]
- “saansi_standard_packages system procedure” on page 1092
- “SQL compliance testing using the SQL Flagger” [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

**Example**

The following statement shows an example of the message that is returned when a disallowed extension is found:

```sql
SELECT SQLFLAGGER('SQL:2003/Package', 'SELECT top 1 dummy_col FROM sys.dummy ORDER BY dummy_col');
```

This statement returns the message '0AW03 Disallowed language extension detected in syntax near 'top' on line 1'.

The following statement returns '00000' because it does not contain any disallowed extensions:

```sql
SELECT SQLFLAGGER('SQL:2003/Package', 'SELECT dummy_col FROM sys.dummy');
```

**SQRT function [Numeric]**

Returns the square root of a number.

**Syntax**

```sql
SQRT( numeric-expression )
```

**Parameters**

- **numeric-expression**  The number for which the square root is to be calculated.

**Returns**

DOUBLE
**Remarks**
This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

**Standards and compatibility**
- **SQL/2008** The SQRT function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions".

**Example**
The following statement returns the value 3.

```sql
SELECT SQRT(9);
```

**STACK_TRACE function [Miscellaneous]**
Returns the information about the stack trace of the current statement.

**Syntax**
```sql
STACK_TRACE()
```

**Returns**
LONG VARCHAR representing the stack trace of the current statement.

**Remarks**
Each line of the returned value contains the qualified procedure name or batch type, followed by the line number of the statement. The last line of the returned value does not end in the new line character. The first line of the stack trace represents the line where the function was invoked. Procedures invoked from hidden procedures return `<hidden>` in the stack trace and not the procedure name.

This function returns the same information as the sa_stack_trace system procedure.

**See also**
- "TRY statement" on page 1022
- "BEGIN statement" on page 523
- "ERROR_LINE function [Miscellaneous]" on page 228
- "ERROR_MESSAGE function [Miscellaneous]" on page 229
- "ERROR_PROCEDURE function [function type]" on page 230
- "ERROR_SQLCODE function [Miscellaneous]" on page 231
- "ERROR_SQLSTATE function [Miscellaneous]" on page 232
- "sa_error_stack_trace system procedure" on page 1136
- "sa_stack_trace system procedure" on page 1250
- "Nested compound statements and exception handlers" [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**
- **SQL/2008** Vendor extension.
STDDEV function [Aggregate]

An alias for STDDEV_SAMP.

See also

- “STDDEV_SAMP function [Aggregate]” on page 373

STDDEV_POP function [Aggregate]

Computes the standard deviation of a population consisting of a numeric-expression, as a DOUBLE.

Syntax 1

```
STDDEV_POP( numeric-expression )
```

Syntax 2

```
STDDEV_POP( numeric-expression ) OVER ( window-spec )
```

`window-spec` : see Syntax 2 instructions in the Remarks section below

Parameters

- **numeric-expression**  The expression whose population-based standard deviation is calculated over a set of rows. The expression is commonly a column name.

Returns

DOUBLE

Remarks

This function converts its argument to DOUBLE, and performs the computation in double-precision floating-point arithmetic.

The population-based standard deviation (s) is computed according to the following formula:

\[
s = \left[ \frac{1}{N} \sum (x_i - \text{mean}(x))^2 \right]^{\frac{1}{2}}
\]

This standard deviation does not include rows where `numeric-expression` is NULL. It returns NULL for a group containing no rows.

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of `window-spec` can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the `window-spec` definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].
For more information about specifying a window specification in an OVER clause, see “Window
definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also

- “WINDOW clause” on page 1051
- “Aggregate functions” on page 143

Standards and compatibility

- SQL/2008 The STDDEV_POP function comprises part of optional SQL /2008 language feature T621, "Enhanced numeric functions". When used as window function, STDDEV_POP comprises part of optional SQL foundation feature T611, "Elementary OLAP operations".

  The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional SQL language feature F561, "Full value expressions". SQL Anywhere also supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

  SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the STDDEV_POP function, combined with an outer reference. For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example

The following statement lists the average and variance in the number of items per order in different time periods:

```sql
SELECT YEAR( ShipDate ) AS Year,
       QUARTER( ShipDate ) AS Quarter,
       AVG( Quantity ) AS Average,
       STDDEV_POP( Quantity ) AS Variance
FROM GROUPO.SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Average</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1</td>
<td>25.775148</td>
<td>14.2794...</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>27.050847</td>
<td>15.0270...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

STDDEV_SAMP function [Aggregate]

Computes the standard deviation of a sample consisting of a numeric-expression, as a DOUBLE.
Syntax 1

\[
\text{STDDEVsamp}(\text{numeric-expression})
\]

Syntax 2

\[
\text{STDDEVsamp}(\text{numeric-expression}) \text{ OVER (window-spec)}
\]

\textit{window-spec} : see Syntax 2 instructions in the Remarks section below

Parameters

- \textit{numeric-expression} The expression whose sample-based standard deviation is calculated over a set of rows. The expression is commonly a column name.

Returns

\text{DOUBLE}

Remarks

This function converts its argument to \text{DOUBLE}, and performs the computation in double-precision floating-point arithmetic.

The standard deviation (s) is computed according to the following formula, which assumes a normal distribution:

\[
s = \left[ \frac{1}{(N - 1)} \times \text{SUM} \left( x_i - \text{mean}(x) \right)^2 \right]^{\frac{1}{2}}
\]

This standard deviation does not include rows where \text{numeric-expression} is \text{NULL}. It returns \text{NULL} for a group containing either 0 or 1 rows.

For more information about the statistical computation performed, see “Mathematical formulas for the aggregate functions” [SQL Anywhere Server - SQL Usage].

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of \text{window-spec} can be specified either in the function syntax (inline), or with a \text{WINDOW} clause in the SELECT statement. See the \textit{window-spec} definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also

- “\text{WINDOW clause}” on page 1051
- “Aggregate functions” on page 143

Standards and compatibility

- \text{SQL/2008} The STDDEV_SAMP function comprises part of optional SQL /2008 language feature T621, "Enhanced numeric functions". When used as window function, STDDEV_SAMP comprises part of optional SQL foundation feature T611, "Elementary OLAP operations".
The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional SQL language feature F561, "Full value expressions". SQL Anywhere also supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the STDDEV_SAMP function, combined with an outer reference. For an example, see “AVG function [Aggregate]” on page 164.

Example

The following statement lists the average and variance in the number of items per order in different time periods:

```sql
SELECT YEAR(ShipDate) AS Year,
      QUARTER(ShipDate) AS Quarter,
      AVG(Quantity) AS Average,
      STDDEV_SAMP(quantity) AS Variance
FROM GROUPO.SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Average</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1</td>
<td>25.775148</td>
<td>14.3218...</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>27.050847</td>
<td>15.0696...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**STR function [String]**

Returns the string equivalent of a number.

**Syntax**

```sql
STR( numeric-expression [, length [, decimal ] ] )
```

**Parameters**

- **numeric-expression** Any approximate numeric (float, real, or double precision) expression between -1E126 and 1E127.
- **length** The number of characters to be returned (including the decimal point, all digits to the right and left of the decimal point, and blanks). The default is 10.
- **decimal** The number of decimal digits to be returned. The default is 0.

**Returns**

VARCHAR
Remarks
If the integer portion of the number cannot fit in the length specified, then the result is a string of the specified length containing all asterisks. For example, the following statement returns ***.

```
SELECT STR( 1234.56, 3 );
```

Note
The maximum length that is supported is 128. Any length that is not between 1 and 128 yields a result of NULL.

See also
- “String functions” on page 152

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement returns a string of six spaces followed by 1235, for a total of ten characters.

```
SELECT STR( 1234.56 );
```

The following statement returns the result 1234.6.

```
SELECT STR( 1234.56, 6, 1 );
```

**STRING function [String]**

Concatenates one or more strings into one large string.

Syntax

```
STRING( string-expression [, ... ] )
```

Parameters

- **string-expression** The string to be evaluated.

If only one argument is supplied, it is converted into a single expression. If multiple arguments are supplied, they are concatenated into a single string.

Returns

- LONG VARCHAR
- LONG NVARCHAR
- LONG BINARY

Remarks

Numeric or date parameters are converted to strings before concatenation. The STRING function can also be used to convert any single expression to a string by supplying that expression as the only parameter.
If all parameters are NULL, STRING returns NULL. If any parameters are non-NULL, then any NULL parameters are treated as empty strings.

See also
● “String functions” on page 152

Standards and compatibility
● SQL/2008   Vendor extension.

Example
The following statement returns the value testing123.

```
SELECT STRING( 'testing', NULL, 123 );
```

### STRTOUUID function [String]

Converts a string value to a unique identifier (UUID or GUID) value.

#### Note
In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type was defined as a user-defined data type and the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values.

In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type and SQL Anywhere carries out conversions as needed. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions.

For more information, see “UNIQUEIDENTIFIER data type” on page 126.

### Syntax

```
STRTOUUID( string-expression )
```

### Parameters

- **string-expression** A string in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

### Returns

UNIQUEIDENTIFIER

### Remarks

Converts a string in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`, where `x` is a hexadecimal digit, to a unique identifier value.

If the string is not a valid UUID string, a conversion error is returned unless the conversion_error option is set to OFF, in which case it returns NULL.

This function is useful for inserting UUID values into a database.
This function supports NCHAR inputs and/or outputs.

Curly braces can be used as the first and last characters in the \textit{string-expression}.

\textbf{See also}

\begin{itemize}
  \item “UUIDTOSTR function [String]” on page 401
  \item “NEWID function [Miscellaneous]” on page 303
  \item “String functions” on page 152
\end{itemize}

\textbf{Standards and compatibility}

\begin{itemize}
  \item SQL/2008 Vendor extension.
\end{itemize}

\textbf{Example}

The following statements are equivalent and return the result 0x6c2b64a93c6f47dc901536b9ed49fec2.

\begin{verbatim}
SELECT STRTOUUID( '6c2b64a9-3c6f-47dc-9015-36b9ed49fec2' );
SELECT STRTOUUID( '{6c2b64a9-3c6f-47dc-9015-36b9ed49fec2}' );
\end{verbatim}

\textbf{STUFF function [String]}

Deletes multiple characters from one string and replaces them with another string.

\textbf{Syntax}

\begin{verbatim}
STUFF( string-expression-1, start, length, string-expression-2 )
\end{verbatim}

\textbf{Parameters}

\begin{itemize}
  \item \textit{string-expression-1} The string to be modified by the STUFF function.
  \item \textit{start} The character position at which to begin deleting characters. The first character in the string is position 1.
  \item \textit{length} The number of characters to delete.
  \item \textit{string-expression-2} The string to be inserted. To delete a portion of a string using the STUFF function, use a replacement string of NULL.
\end{itemize}

\textbf{Returns}

\texttt{LONG NVARCHAR}

\textbf{Remarks}

This function supports NCHAR inputs and/or outputs.

\textbf{See also}

\begin{itemize}
  \item “INSERTSTR function [String]” on page 273
  \item “String functions” on page 152
\end{itemize}
Standards and compatibility
- **SQL/2008** Vendor extension.

Example
The following statement returns the value chocolate pie.

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' );
```

**SUBSTRING function [String]**
Returns a substring of a string.

Syntax
```
{ SUBSTRING | SUBSTR } ( string-expression, start [, length ] )
```

Parameters
- **string-expression** The string from which a substring is to be returned.
- **start** The start position of the substring to return, in characters.
- **length** The length of the substring to return, in characters. If *length* is specified, the substring is restricted to that length.

Returns
- LONG BINARY
- LONG VARCHAR
- LONG NVARCHAR

Remarks
The behavior of this function depends on the setting of the ansi_substring database option. When the ansi_substring option is set to On (the default), the behavior of the SUBSTRING function corresponds to ANSI/ISO SQL/2008 behavior. The behavior is as follows:

<table>
<thead>
<tr>
<th>ansi_substring option setting</th>
<th>start value</th>
<th>length value</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>The first character in the string is at position 1. A negative or zero start offset is treated as if the string were padded on the left with non-characters.</td>
<td>A positive <em>length</em> specifies that the substring ends <em>length</em> characters to the right of the starting position.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A negative <em>length</em> returns an error.</td>
</tr>
</tbody>
</table>
If string-expression is of binary data type, the SUBSTRING function behaves as BYTE_SUBSTR.

To obtain characters at the end of a string, use the RIGHT function.

This function supports NCHAR inputs and/or outputs. Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics.

### See also

- “BYTE_SUBSTR function [String]” on page 172
- “LEFT function [String]” on page 282
- “RIGHT function [String]” on page 350
- “CHARINDEX function [String]” on page 177
- “String functions” on page 152

### Standards and compatibility

- **SQL/2008**  
  SUBSTRING is a core feature of the SQL/2008 standard. The standard's implementation differs slightly from the SQL Anywhere implementation: in the standard, SUBSTRING is defined with three parameters using the keywords FROM and FOR, neither of which are required by SQL Anywhere.

### Example

The following table shows the values returned by the SUBSTRING function.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBSTRING( 'front yard', 1, 4 )</td>
<td>fron</td>
</tr>
<tr>
<td>SUBSTRING( 'back yard', 6, 4 )</td>
<td>yard</td>
</tr>
<tr>
<td>SUBSTR( 'abcdefg', 0, -2 )</td>
<td>Returns an error if ansi_substring is On</td>
</tr>
<tr>
<td>SUBSTR( 'abcdefg', -2, 2 )</td>
<td>Returns an empty string if ansi_substring is On</td>
</tr>
</tbody>
</table>
SUM function [Aggregate]

Returns the total of the specified expression for each group of rows.

Syntax 1

```
SUM( [ ALL | DISTINCT ] expression )
```

Syntax 2

```
SUM( [ ALL ] expression ) OVER ( window-spec )
```

*window-spec*: see Syntax 2 instructions in the Remarks section below

Parameters

- **[ ALL ] expression**  The name of the expression to be summed. This is commonly a column name.
- **DISTINCT expression**  Computes the sum of the unique values of *expression* within each group.

Returns

- INTEGER
- DOUBLE
- NUMERIC

Remarks

Rows where the specified expression is NULL are not included.

Returns NULL for a group containing no rows.

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of *window-spec* can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the *window-spec* definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

This function can generate an overflow error, resulting in an error being returned. You can use the CAST function on *numeric-expression* to avoid the overflow error. See “CAST function [Data type conversion]” [UltraLite - Database Management and Reference].

See also

- “WINDOW clause” on page 1051
- “COUNT function [Aggregate]” on page 192
- “AVG function [Aggregate]” on page 164

Standards and compatibility

- **SQL/2008**  Core feature. When used as a window function (Syntax 2), SUM comprises part of optional SQL/2008 language feature T611, "Basic OLAP operations".
The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional SQL language feature F561, "Full value expressions". SQL Anywhere also supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the SUM function, combined with an outer reference. See “Troubleshooting: Aggregate functions and outer references” [SQL Anywhere 16 - Changes and Upgrading].

For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example
The following statement returns the value 3749146.740.

```
SELECT SUM( Salary )
FROM GROUPO.Employees;
```

**SUSER_ID function [System]**

Returns the numeric user ID for the specified user name.

**Syntax**

```
SUSER_ID ([ user-name ]
```

**Parameters**

- **user-name** The user name for the user ID you are searching for.

**Returns**

VARCHAR

**Remarks**

If you do not specify user-name, the ID of the current user is returned.

This function is provided for compatibility with other vendors. You can also use the USER_ID function, which does exactly the same thing.

**See also**

- “SUSER_NAME function [System]” on page 383
- “USER_ID function [System]” on page 400

**Standards and compatibility**

- **SQL/2008** Transact-SQL extension.
Example

The following statement returns the ID for the GROUPO user.

```
SELECT SUSER_ID( 'GROUPO' );
```

**SUSER_NAME function [System]**

Returns the user name for the specified user ID.

**Syntax**

```
SUSER_NAME( [ user-id ] )
```

**Parameters**

- **user-id**  
The user ID of the user you are searching for.

**Returns**

`VARCHAR`

**Remarks**

If you do not specify `user-id`, the user name of the current user is returned.

This function is provided for compatibility with other vendors. You can also use the USER_NAME function, which does exactly the same thing.

**See also**

- “SUSER_ID function [System]” on page 382
- “USER_NAME function [System]” on page 400

**Standards and compatibility**

- **SQL/2008**  
  Transact-SQL extension.

**Example**

The following statement returns the user name for a user with ID 101.

```
SELECT SUSER_NAME( 101 );
```

**SWITCHOFFSET function [Date and time]**

Returns a TIMESTAMP WITH TIME ZONE value that is converted from its original time zone offset to the specified time zone offset.

**Syntax**

```
SWITCHOFFSET( tmz-expression, time-zone-offset )
```
Parameters

- **tmz-expression**  The TIMESTAMP WITH TIME ZONE value to be converted.
- **time-zone-offset**  The time zone offset of the result. The value can be an integer representing the minutes before or after Coordinated Universal Time (UTC), a string in the form { + | - } hh:nn, or Z for the Zulu Time Zone. The Zulu Time Zone is the same time zone as UTC.

Returns

TIMESTAMP WITH TIME ZONE

See also

- “TIMESTAMP WITH TIME ZONE data type” on page 123
- “SYSDATETIMEOFFSET function [Date and time]” on page 384

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example changes a time zone offset value from -04:00 hours to -07:00 hours. The value returned is 2009-04-03 11:45:12.123-07:00.

```
SELECT CAST ( '2009-04-03 14:45:12.123-04:00' AS datetimeoffset ) AS EDT,
SWITCHOFFSET( EDT,'-07:00' ) AS PDT;
```

**SYSDATETIMEOFFSET function [Date and time]**

Returns the current date, time, and time zone offset of the database server using the system clock.

Syntax

```
SYSDATETIMEOFFSET ( )
```

Returns

TIMESTAMP WITH TIME ZONE

See also

- “TIMESTAMP WITH TIME ZONE data type” on page 123
- “SWITCHOFFSET function [Date and time]” on page 383

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example returns the current date and time and the time zone offset of the database server.

```
SELECT SYSDATETIMEOFFSET ( );
```
The following example converts the SYSDATETIMEOFFSET value to the time zone of the client computer.

```sql
SELECT SWITCHOFFSET ( SYSDATETIMEOFFSET ( ),
  CAST( connection_property ('TimeZoneAdjustment') AS INT ));
```

### TAN function [Numeric]

Returns the tangent of a number.

**Syntax**

```sql
TAN( numeric-expression )
```

**Parameters**

- `numeric-expression` An angle, in radians.

**Returns**

DOUBLE

**Remarks**

The ATAN and TAN functions are inverse operations.

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

**See also**

- “COS function [Numeric]” on page 191
- “SIN function [Numeric]” on page 362

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statement returns the value of the tan of 0.52.

```sql
SELECT TAN( 0.52 );
```

### TEXTPTR function [Text and image]

Returns a 16-byte binary pointer to the specified column. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

**Syntax**

```sql
TEXTPTR( column-name )
```
Parameters

- **column-name** The name of a column containing CHAR, NCHAR, or BINARY data.

Returns

BINARY

Remarks

This function is included for Transact-SQL compatibility.

Standards and compatibility

- **SQL/2008** Vendor extension.

Example

The following embedded SQL example uses TEXTPTR to locate the Description column associated with ProductID 500 in the MarketingInformation table.

The text pointer is stored in the variable `txtptr` and supplied as a parameter to the READTEXT statement which returns 55 bytes, starting at column offset 181.

```sql
EXEC SQL BEGIN DECLARE SECTION;
char            hostvar[100];
EXEC SQL END DECLARE SECTION;

EXEC SQL create variable txtptr binary(16);
EXEC SQL set txtptr =
    ( SELECT textptr(Description)
    FROM GROUPO.MarketingInformation
    WHERE ProductID = '500' );
EXEC SQL PREPARE S1 FROM
    'READTEXT GROUPO.MarketingInformation.Description txtptr 181 55';
EXEC SQL EXECUTE S1 INTO :hostvar;
printf( "hostvar: %s\n", hostvar );
```

READTEXT returns the following string.

```
Lightweight 100% organically grown cotton construction.
```

**TO_CHAR function [String]**

Converts character data from any supported character set into the CHAR character set for the database.

Syntax

```
TO_CHAR( string-expression [, source-charset-name ] )
```

Parameters

- **string-expression** The string to be converted.

- **source-charset-name** The character set of the string.
Returns
LONG VARCHAR

Remarks
If source-charset-name is specified, then this function is equivalent to:

```sql
CAST( CSCONVERT( CAST( string-expression AS BINARY ),
    'db_charset', source-charset-name )
AS CHAR );
```

For more information about db_charset, see “CSCONVERT function [String]” on page 198.

If source-charset-name is not specified, then this function is equivalent to:

```sql
CAST( string-expression AS CHAR );
```

See also
- “Recommended character sets and collations” [SQL Anywhere Server - Database Administration]
- “CONNECTION_EXTENDED_PROPERTY function [String]” on page 184
- “CSCONVERT function [String]” on page 198
- “NCHAR function [String]” on page 302
- “TO_NCHAR function [String]” on page 387
- “UNICODE function [String]” on page 397
- “UNISTR function [String]” on page 398

Standards and compatibility
- SQL/2008  Vendor extension.

Example
If you have a BINARY value containing data in the cp850 character set, the following statement converts
the data to the CHAR character set and data type:

```sql
SELECT TO_CHAR( 'cp850_data', 'cp850' );
```

TO_NCHAR function [String]
Converts character data from any supported character set into the NCHAR character set.

Syntax
```
TO_NCHAR( string-expression [, source-charset-name ] )
```

Parameters
- **string-expression**  The string to be converted
- **source-charset-name**  The character set of the string.

Returns
LONG NVARCHAR
Remarks

If \texttt{source-charset-name} is specified then this function is equivalent to:

\begin{verbatim}
CAST( CSCONVERT( CAST( string-expression AS BINARY ),
'nchar_charset', source-charset-name )
AS NCHAR );
\end{verbatim}

For more information about \texttt{nchar_charset}, see “CSCONVERT function [String]” on page 198.

If \texttt{source-charset-name} is not provided then this function is equivalent to:

\begin{verbatim}
CAST( string-expression AS NCHAR );
\end{verbatim}

See also

- “Recommended character sets and collations” [SQL Anywhere Server - Database Administration]
- “CONNECTION_EXTENDED_PROPERTY function [String]” on page 184
- “CSCONVERT function [String]” on page 198
- “NCHAR function [String]” on page 302
- “TO_CHAR function [String]” on page 386
- “UNICODE function [String]” on page 397
- “UNISTR function [String]” on page 398

Standards and compatibility

- SQL/2008 Vendor extension.

Example

If you have a BINARY value containing data in the cp850 character set, the following example to converts the data to the NCHAR character set and data type:

\begin{verbatim}
SELECT TO_NCHAR( 'cp850_data', 'cp850' );
\end{verbatim}

\section*{TODATETIMEOFFSET function [Date and time]}

Converts a TIMESTAMP value to a TIME STAMP WITH TIME ZONE value using the specified time zone offset.

\subsection*{Syntax}

\texttt{TODATETIMEOFFSET( timestamp-expression, time-zone-offset )}

\subsection*{Parameters}

- \texttt{timestamp-expression} The TIMESTAMP expression to be converted.

- \texttt{time-zone-offset} The time zone offset. The value can be an INTEGER representing minutes before or after UTC, a VARCHAR string in the form of { + | - }hh:nn, or the string "Z" for the Zulu Time Zone. The Zulu Time Zone is the same time zone as UTC.

\subsection*{Returns}

TIMESTAMP WITH TIME ZONE
See also

- “TIMESTAMP WITH TIME ZONE data type” on page 123

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example converts a TIMESTAMP value to a TIMESTAMP WITH TIME ZONE value.

```
SELECT CAST('2009-04-03 14:45:12.123' AS TIMESTAMP) AS orig,
       TODATETIMEOFFSET (orig,'+11:00');
```

TODAY function [Date and time]

Returns the current date as a DATE value.

Syntax

```
TODAY([ * ])
```

Returns

DATE

Remarks

TODAY(*) and TODAY() are semantically equivalent. TODAY is equivalent to the CURRENT DATE special value.

Each instance of the TODAY function in a request is evaluated at most once. Multiple instances of TODAY in the same request may or may not share the identical DATE value.

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statements return the current day according to the system clock.

```
SELECT TODAY( * );
SELECT CURRENT DATE;
```

TRACEBACK function [Miscellaneous]

Returns statements on the stack of the most recent exception (error) that occurred during a stored procedure, trigger, or custom function execution.

Syntax

```
TRACEBACK([ * ])
```
**Returns**

LONG VARCHAR

**Remarks**

The returned call stack is annotated with the object names and line numbers. Statements from stored procedures with HIDDEN definitions are not written to the stack.

The statements in the TRACEBACK are generated from the database server representation of the statements in the stored procedures, which may not precisely match the text in the procedure definition.

TRACEBACK(*) and TRACEBACK() are semantically equivalent.

This function is useful for debugging procedures and triggers, particularly those that are written in the Transact-SQL dialect.

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

To use the TRACEBACK function, execute the following statement after an error occurs while executing a procedure:

```
SELECT TRACEBACK( * );
```

Output similar to the following output is returned after using the TRACEBACK function:

```
"user1"."proc1" : 10 : set ret_val = in_val / (in_val - in_val)
"user2"."proc2" : 5 : <hidden>
"user3"."proc1" : 7 : call user2.proc2( 10 )
```

**TRACED_PLAN function [Miscellaneous]**

Generates a graphical plan for a query using tracing data in Sybase Central.

**Syntax**

```
TRACED_PLAN( logging_session_id, query_id )
```

**Parameters**

- **logging_session_id**  Combined with query_id, this INTEGER parameter identifies a row from the sa_diagnostic_query table for which to generate the plan.

- **query_id**  Combined with logging_session_id, this INTEGER parameter identifies a row from the sa_diagnostic_query table for which to generate the plan.

**Returns**

LONG VARCHAR
Remarks
   This function is for use by Sybase Central.

See also
   ● “sa_diagnostic_query table” on page 1076

Standards and compatibility
   ● SQL/2008 Vendor extension.

TRANSACTSQL function [Miscellaneous]
   Rewrites a Watcom SQL statement in the Transact-SQL dialect.

Syntax
   TRANSACTSQL( sql-statement-string )

Parameters
   ● sql-statement-string The SQL statement that is to be rewritten in Transact-SQL.

Returns
   LONG VARCHAR

See also
   ● “SQLDIALECT function [Miscellaneous]” on page 368
   ● “WATCOMSQL function [Miscellaneous]” on page 406

Standards and compatibility
   ● SQL/2008 Vendor extension.

Example
   The following statement returns the string 'SELECT EmployeeName=empl_name FROM Employees'.

   SELECT TRANSACTSQL( 'SELECT empl_name as EmployeeName FROM GROUP0.Employees' ) FROM dummy;

TREAT function [Data type conversion]
   The TREAT function allows you to change the declared type of a geometry expression to a subtype. This function is for use with spatial data.

Syntax
   TREAT( geometry-expression AS subtype )
### Parameters

- **geometry-expression** The expression to be converted.
- **subtype** The target subtype to convert `geometry-expression` into.

### Returns

Depends on the data type requested.

### Remarks

The TREAT function can only be used on geometries.

If the dynamic type of the expression is not a subtype of the target data type, an error is returned. The CAST function can also be used to change the declared type of a geometry expression. However, the CAST function allows changes outside of the subtype hierarchy. For example, CAST can be used to convert a point to a multipoint. These types of conversions may change the dynamic type of an expression in unexpected ways, so TREAT is preferable when moving from a supertype to a subtype. The TREAT function also executes more efficiently than the CAST function.

### See also

- “Using the TREAT expression for subtypes” [SQL Anywhere Server - Spatial Data Support]
- “CAST function [Data type conversion]” on page 174

### Standards and compatibility

- **SQL/2008** Vendor extension.

### Examples

Execute the following in Interactive SQL to create a table and load two values into it:

```sql
DROP TABLE IF EXISTS treatExample;
CREATE TABLE treatExample( pk INT PRIMARY KEY, geo ST_Geometry );
INSERT INTO treatExample VALUES(0, NEW ST_Point(3,4) );
INSERT INTO treatExample VALUES(1, NEW ST_MultiPoint( new ST_Point( 5,
6 ) ) );
```

The following query returns an error. See “Type '%1' has no method named '%2' (near '%3')” [Error Messages].

```sql
SELECT geo.ST_X() FROM treatExample T WHERE pk = 0;
```

The following query succeeds:

```sql
SELECT TREAT( geo AS ST_Point ).ST_X() FROM treatExample WHERE pk = 0;
```

The following query returns an error. See “Type '%1' has no method named '%2' (near '%3')” [Error Messages].

```sql
SELECT TREAT( geo AS ST_Point ).ST_X() FROM treatExample T WHERE pk = 0;
```

The following query succeeds because a CAST statement is used instead of a TREAT statement:

```sql
SELECT CAST( geo AS ST_Point ) FROM treatExample WHERE pk = 1;
```
**TRIM function [String]**

Removes leading and trailing blanks from a string.

**Syntax**

```
TRIM( string-expression )
```

**Parameters**

- `string-expression` The string to be trimmed.

**Returns**

- VARCHAR
- NVARCHAR
- LONG VARCHAR
- LONG NVARCHAR

**Remarks**

This function supports NCHAR inputs and/or outputs.

**See also**

- “LTRIM function [String]” on page 291
- “RTRIM function [String]” on page 356
- “String functions” on page 152

**Standards and compatibility**

- **SQL/2008** The TRIM function is a SQL/2008 core feature.

SQL Anywhere does not support the additional parameters `trim specification` and `trim character`, as defined in SQL/2008. The SQL Anywhere implementation of TRIM corresponds to a TRIM specification of BOTH.

For the other TRIM specifications defined by the SQL/2008 standard (LEADING and TRAILING), SQL Anywhere supplies the LTRIM and RTRIM functions respectively.

**Example**

The following statement returns the value chocolate with no leading or trailing blanks.

```
SELECT TRIM( '   chocolate   ' );
```

**TRIM_ARRAY function [Composite]**

Returns an implicitly bounded array that consists of a specified number of elements in an array.

**Syntax**

```
TRIM_ARRAY( array-expression, integer-expression )
```
Parameters

- **array-expression** The array to trim.
- **integer-expression** The number of elements that the resulting array should contain. The resulting array contains the first integer-expression number of elements in the array-expression.

If the integer-expression is zero, the resulting array is empty. If the integer-expression is less than zero, or is greater than the cardinality of the array, then an error is generated.

Returns

ARRAY

Remarks

If either argument to TRIM_ARRAY is NULL, then the result is NULL.

See also

- “ARRAY_AGG function [Aggregate]” on page 159
- “ARRAY_MAX_CARDINALITY function [Composite]” on page 160
- “CARDINALITY function [Composite]” on page 173

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example illustrates how to use the TRIM_ARRAY function to create an array that consists of the first two elements in an array that contains four elements:

```sql
DECLARE UntrimmedArray ARRAY( 4 ) OF INT;
DECLARE TrimmedArray ARRAY( 2 ) OF INT;
SET UntrimmedArray = ARRAY( 1, 2, 3, 4 );
SET TrimmedArray = TRIM_ARRAY( UntrimmedArray, 2 );
```

TRUNCNUM function [Numeric]

Truncates a number at a specified number of places after the decimal point.

Syntax

```sql
{ TRUNCNUM | TRUNCATE } ( numeric-expression, integer-expression )
```

Parameters

- **numeric-expression** The number to be truncated. This argument may be of type NUMERIC or DOUBLE.
- **integer-expression** A positive integer specifies the number of significant digits to the right of the decimal point at which to round. A negative value specifies the number of significant digits to the left of the decimal point at which to round.
Returns
NUMERIC or DOUBLE

Remarks
You should use the TRUNCNUM function, not the TRUNCATE function, when truncating numbers.

Use of the TRUNCATE function is not recommended because the word truncate is a keyword, and therefore requires you to either set the quoted_identifier option to OFF, or put quotes around the word TRUNCATE.

See also
● “ROUND function [Numeric]” on page 351
● “quoted_identifier option” [SQL Anywhere Server - Database Administration]

Standards and compatibility
● SQL/2008 Vendor extension.

Example
The following statement returns the value 600.

```
SELECT TRUNCNUM( 655, -2 );
```

The following statement: returns the value 655.340.

```
SELECT TRUNCNUM( 655.348, 2 );
```

**TSEQUAL function [System] (deprecated)**

Compares two TIMESTAMP values and returns whether they are the same.

Syntax
```
TSEQUAL ( timestamp-expression-1, timestamp-expression-2 )
```

Parameters
● `timestamp-expression-1` A TIMESTAMP value.
● `timestamp-expression-2` A TIMESTAMP value.

Returns
BIT

Remarks
The TSEQUAL function can only be used in a WHERE clause and is most commonly used as part of an UPDATE statement.
Although the TSEQUAL function can be used to compare two ordinary TIMESTAMP values, the purpose of TSEQUAL is to determine whether or not a row has been modified by another connection by comparing two special Transact-SQL TIMESTAMP values.

In a single-row UPDATE statement using TSEQUAL, if timestamp-expression-1 is equal to timestamp-expression-2 and one of these values refers to a column declared with DEFAULT TIMESTAMP and the other refers to the value of the column when the row was last fetched, then the row has not changed since it was fetched and TSEQUAL returns TRUE. If the row was changed by another user, its timestamp has been modified and the TSEQUAL function returns FALSE. If the TSEQUAL function returns FALSE in this situation, the UPDATE is not performed. The application can determine that no rows were updated by examining the number of rows affected, for example by using @@rowcount. If no rows were affected, the application can assume that the row was modified by another user and that it needs to be re-fetched.

See also

- “The data type of a TIMESTAMP column” [SQL Anywhere Server - SQL Usage]
- “TIMESTAMP special value” on page 76
- “The special Transact-SQL TIMESTAMP column and data type” [SQL Anywhere Server - SQL Usage]
- “UPDATE statement” on page 1037

Standards and compatibility

- SQL/2008 Vendor extension.

Example

Suppose you create a TIMESTAMP column Products.LastUpdated to store the timestamp for the last time the row was updated. The following example uses the TSEQUAL function to change a row value. An update is applied to the row only when the row has not been changed since it was last fetched.

```sql
SELECT LastUpdated into old_ts_value
FROM GROUPO.Products
WHERE ID = '300';

UPDATE GROUPO.Products
SET Color = 'Yellow'
WHERE ID = '300'
AND TSEQUAL( LastUpdated, old_ts_value );
```

**UCASE function [String]**

Converts all characters in a string to uppercase.

**Syntax**

`UCASE( string-expression )`

**Parameters**

- `string-expression` The string to be converted to uppercase.
Returns

CHAR, VARCHAR, LONG VARCHAR, NCHAR, NVARCHAR, or LONG NVARCHAR corresponding to the data type of the argument.

Remarks

This function is identical to the UPPER function.

See also

- “UPPER function [String]” on page 399
- “LCASE function [String]” on page 281
- “String functions” on page 152

Standards and compatibility

- SQL/2008 Vendor extension. The UPPER function is SQL/2008 compliant.

Example

The following statement returns the value CHOCOLATE.

```
SELECT UCASE('ChocoLate');
```

**UNICODE function [String]**

Returns an integer containing the Unicode code point of the first character in the string, or NULL if the first character is not a valid encoding.

Syntax

```
UNICODE(nchar-string-expression)
```

Parameters

- `nchar-string-expression` The NCHAR string whose first character is to be converted to an integer.

Returns

INT

See also

- “CONNECTION_EXTENDED_PROPERTY function [String]” on page 184
- “NCHAR function [String]” on page 302
- “TO_CHAR function [String]” on page 386
- “TO_NCHAR function [String]” on page 387
- “UNISTR function [String]” on page 398

Standards and compatibility

- SQL/2008 Vendor extension.
Example
The following example returns the integer 65536:

```sql
SELECT UNICODE(UNISTR( 'Ā00data' ));
```

**UNISTR function [String]**
Converts a string containing characters and Unicode escape sequences to an NCHAR string.

**Syntax**

```
UNISTR( string-expression )
```

**Parameters**
- `string-expression` The string to be converted.

**Returns**
- NVARCHAR
- LONG NVARCHAR

**Remarks**
The UNISTR function allows the use of Unicode characters that cannot be represented in the CHAR character set used by the SQL statement. For example, in an English environment, the UNISTR function could be used to include Chinese characters.

The UNISTR function offers similar functionality to the N" constant, however the UNISTR function allows Unicode characters and characters from the CHAR character set, whereas the N" constant only allows characters from the CHAR character set.

The `string-expression` contains characters and Unicode escape sequences. The Unicode escape sequences are of the form \uXXXX or \uXXXXXX, where each X is a hexadecimal digit. The UNISTR function converts each character and each Unicode escape sequence to the corresponding Unicode character.

If a 6-digit Unicode escape sequence is used, its value must not exceed 10FFFF, the largest Unicode code point. A sequence such as \u234567 is not a 6-digit Unicode escape sequence. It is the 4-digit sequence \u2345 followed by the characters 6 and 7.

If two adjacent Unicode escape sequences form a UTF-16 surrogate pair, they are combined into one Unicode character in the output.

**See also**
- “CONNECTION_EXTENDED_PROPERTY function [String]” on page 184
- “NCHAR function [String]” on page 302
- “TO_CHAR function [String]” on page 386
- “TO_NCHAR function [String]” on page 387
- “UNICODE function [String]” on page 397
Standards and compatibility

- SQL/2008  Vendor extension.

Examples

The following example returns the string Hello.

```sql
SELECT UNISTR( 'Hel\u006c\u006F' );
```

The following example combines the UTF-16 surrogate pair D800-DF02 into the Unicode code point 10302.

```sql
SELECT UNISTR( '\uD800\uDF02' );
```

The example is equivalent to the previous one.

```sql
SELECT UNISTR( '\u010302' );
```

UPPER function [String]

Converts all characters in a string to uppercase.

Syntax

```sql
UPPER( string-expression )
```

Parameters

- `string-expression`  The string to be converted to uppercase.

Returns

CHAR, VARCHAR, LONG VARCHAR, NCHAR, NVARCHAR, or LONG NVARCHAR corresponding to the data type of the argument.

Remarks

This function is identical to the UCASE function.

See also

- “UCASE function [String]” on page 396
- “LCASE function [String]” on page 281
- “LOWER function [String]” on page 290
- “String functions” on page 152

Standards and compatibility

- SQL/2008  The UPPER function is a core feature of the SQL/2008 standard.

Example

The following statement returns the value CHOCOLATE.

```sql
SELECT UPPER( 'ChocoLate' );
```
USER_ID function [System]

Returns the numeric user ID for the specified user name.

Syntax

USER_ID( [ user-name ] )

Parameters

- **user-name**  
The user name for the numeric user ID you are searching for.

Returns

INTEGER

Remarks

If you do not specify *user-name*, the numeric user ID of the current user is returned.

See also

- “USER_NAME function [System]” on page 400
- “SUSER_ID function [System]” on page 382

Standards and compatibility

- SQL/2008  
Vendor extension.

Example

The following statement returns the numeric user ID for user name GROUPO.

```
SELECT USER_ID( 'GROUPO' );
```

USER_NAME function [System]

Returns the user name for the specified user ID.

Syntax

USER_NAME( [ user-id ] )

Parameters

- **user-id**  
The user ID of the user you are searching for.

Returns

VARCHAR

Remarks

If you do not specify *user-id*, the user name of the current user is returned.
See also
- “USER_ID function [System]” on page 400
- “SUSER_NAME function [System]” on page 383

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement returns the user name for user ID 101.

```sql
SELECT USER_NAME( 101 );
```

**UUIDTOSTR function [String]**
Converts a unique identifier value (UUID, also known as GUID) to a string value.

**Note**
In databases created before version 9.0.2, the UNIQUEIDENTIFIER data type was defined as a user-defined data type and the STRTOUUID and UUIDTOSTR functions were needed to convert between binary and string representations of UUID values.

In databases created using version 9.0.2 or later, the UNIQUEIDENTIFIER data type was changed to a native data type and SQL Anywhere carries out conversions as needed. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions.

For more information, see “UNIQUEIDENTIFIER data type” on page 126.

**Syntax**

```
UUIDTOSTR( uuid-expression )
```

**Parameters**
- **uuid-expression** A unique identifier value.

**Returns**
VARCHAR

**Remarks**
Converts a unique identifier to a string value in the format xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, where x is a hexadecimal digit. If the binary value is not a valid uniquedentifier, NULL is returned.

This function is useful for viewing a UUID value.
See also

- "NEWID function [Miscellaneous]" on page 303
- "STRTOUUID function [String]" on page 377
- “String functions” on page 152

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement creates a table mytab with two columns. Column pk has a unique identifier data type, and column c1 has an integer data type. It then inserts two rows with the values 1 and 2 respectively into column c1.

```sql
CREATE TABLE mytab(
    pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    c1 INT );
INSERT INTO mytab( c1 ) values ( 1 );
INSERT INTO mytab( c1 ) values ( 2 );
```

Executing the following SELECT statement returns all the data in the newly created table.

```sql
SELECT * FROM mytab;
```

You will see a two-column, two-row table. The value displayed for column pk will be binary values.

To convert the unique identifier values into a readable format, execute the following statement:

```sql
SELECT UUIDTOSTR(pk), c1 FROM mytab;
```

The UUIDTOSTR function is not needed for databases created with version 9.0.2 or later.

**VAR_POP function [Aggregate]**

Computes the statistical variance of a population consisting of a numeric-expression, as a DOUBLE.

**Syntax 1**

```sql
VAR_POP( numeric-expression )
```

**Syntax 2**

```sql
VAR_POP( numeric-expression ) OVER ( window-spec )
```

*window-spec*: see Syntax 2 instructions in the Remarks section below

**Parameters**

- **numeric-expression**  The expression whose population-based variance is calculated over a set of rows. The expression is commonly a column name.

**Returns**

- **DOUBLE**
Remarks

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

The population-based variance (s^2) of numeric-expression (x) is computed according to the following formula:

\[ s^2 = \frac{1}{N} \sum (x_i - \text{mean}(x))^2 \]

This variance does not include rows where numeric-expression is NULL. It returns NULL for a group containing no rows.

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of window-spec can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the window-spec definition for the WINDOW clause.

For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also

- “Aggregate functions” on page 143
- “WINDOW clause” on page 1051

Standards and compatibility

- SQL/2008 The VAR_POP function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions". When used as window function, VAR_POP comprises part of optional SQL foundation feature T611, "Elementary OLAP operations".

The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional SQL language feature F561, "Full value expressions". SQL Anywhere also supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the VAR_POP function, combined with an outer reference. For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example

The following statement lists the average and variance in the number of items per order in different time periods:

```sql
SELECT YEAR( ShipDate ) AS Year,
       QUARTER( ShipDate ) AS Quarter,
```
AVG(Quantity) AS Average,
VAR_POP(quantity) AS Variance
FROM GROUPO.SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Average</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1</td>
<td>25.775148</td>
<td>203.9021...</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>27.050847</td>
<td>225.8109...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**VAR_SAMP function [Aggregate]**

Computes the statistical variance of a sample consisting of a numeric-expression, as a DOUBLE.

**Syntax 1**

```
VAR_SAMP( numeric-expression )
```

**Syntax 2**

```
VAR_SAMP( numeric-expression ) OVER ( window-spec )
```

*window-spec*: see Syntax 2 instructions in the Remarks section below

**Parameters**

- *numeric-expression*  The expression whose sample-based variance is calculated over a set of rows. The expression is commonly a column name.

**Returns**

DOUBLE

**Remarks**

This function converts its argument to DOUBLE, performs the computation in double-precision floating-point arithmetic, and returns a DOUBLE as the result.

The variance \( (s^2) \) of *numeric-expression* \((x)\) is computed according to the following formula, which assumes a normal distribution:

\[
s^2 = \frac{1}{( N - 1 )} \times \text{SUM}( x_i - \text{mean}( x ) )^2
\]

This variance does not include rows where *numeric-expression* is NULL. It returns NULL for a group containing either 0 or 1 rows.

Syntax 2 represents usage as a window function in a SELECT statement. As such, elements of *window-spec* can be specified either in the function syntax (inline), or with a WINDOW clause in the SELECT statement. See the *window-spec* definition for the WINDOW clause.
For more information about using window functions in SELECT statements, including working examples, see “Window functions” [SQL Anywhere Server - SQL Usage].

For more information about specifying a window specification in an OVER clause, see “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage].

See also

- “Aggregate functions” on page 143
- “VARIANCE function [Aggregate]” on page 406
- “WINDOW clause” on page 1051

Standards and compatibility

- **SQL/2008** The VAR_SAMP function comprises part of optional SQL/2008 language feature T621, "Enhanced numeric functions". When used as window function, VAR_SAMP comprises part of optional SQL foundation feature T611, "Elementary OLAP operations". The VARIANCE syntax is a vendor extension.

  The ability to specify DISTINCT over an expression that is not a column reference comprises part of optional SQL language feature F561, "Full value expressions". SQL Anywhere also supports SQL/2008 language feature F441, "Extended set function support", which permits operands of aggregate functions to be arbitrary expressions possibly including outer references to expressions in other query blocks that are not column references.

  SQL Anywhere does not support optional SQL/2008 feature F442, "Mixed column references in set functions". SQL Anywhere does not permit the arguments of an aggregate function to include both a column reference from the query block containing the VAR_SAMP function, combined with an outer reference. For an example, see “AVG function [Aggregate]” [UltraLite - Database Management and Reference].

Example

The following statement lists the average and variance in the number of items per order in different time periods:

```sql
SELECT YEAR( ShipDate ) AS Year,
  QUARTER( ShipDate ) AS Quarter,
  AVG( Quantity ) AS Average,
  VAR_SAMP( quantity ) AS Variance
FROM GROUPO.SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Quarter</th>
<th>Average</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1</td>
<td>25.775148</td>
<td>205.1158...</td>
</tr>
<tr>
<td>2000</td>
<td>2</td>
<td>27.050847</td>
<td>227.0939...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
VAREXISTS function [Miscellaneous]

Returns 1 if a user-defined variable has been created or declared with a given name. Returns 0 if no such variable has been created.

Syntax

VAREXISTS( variable-name-string )

Parameters

- **variable-name-string** The variable name to be tested, as a string.

Returns

INT

See also

- “CREATE VARIABLE statement” on page 722
- “DECLARE statement” on page 736
- “IF statement” on page 850

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following IF statement creates a variable with a name start_time if one is not already created or declared. The variable can then be used safely.

```
IF VAREXISTS( 'start_time' ) = 0 THEN
  CREATE VARIABLE start_time TIMESTAMP;
END IF;
SET start_time = current timestamp;
```

VARIANCE function [Aggregate]

An alias for VAR_SAMP.

See also

- “VAR_SAMP function [Aggregate]” on page 404

WATCOMSQL function [Miscellaneous]

Rewrites a Transact-SQL statement in the Watcom SQL dialect. This can be useful when converting existing Adaptive Server Enterprise stored procedures into Watcom SQL syntax.

Syntax

WATCOMSQL( sql-statement-string )
Parameters
- sql-statement-string  The SQL statement that the function rewrites into the Watcom SQL dialect.

Returns
LONG VARCHAR

See also
- “SQLDIALECT function [Miscellaneous]” on page 368
- “TRANSACTSQL function [Miscellaneous]” on page 391

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement returns the string 'SELECT Surname as last_name FROM Employees':

```
SELECT WATCOMSQL( 'SELECT last_name = Surname FROM GROUPO.Employees' ) FROM dummy;
```

**WEEKS function [Date and time]**

Manipulates a TIMESTAMP or returns the number of weeks between two TIMESTAMP values. See the Remarks section below.

Syntax 1
```
WEEKS( timestamp-expression )
```

Syntax 2
```
WEEKS( timestamp-expression, timestamp-expression )
```

Syntax 3
```
WEEKS( timestamp-expression, integer-expression )
```

Parameters
- timestamp-expression  A date and time value of type TIMESTAMP.

- integer-expression   The number of weeks to be added to timestamp-expression. If integer-expression is negative, the appropriate number of weeks is subtracted from timestamp-expression. If you supply an integer-expression, timestamp-expression must be explicitly cast as a DATE or TIMESTAMP.

Returns
INTEGER with Syntax 1 or Syntax 2.

TIMESTAMP with Syntax 3.
Remarks

Given a single date (Syntax 1), the WEEKS function returns the number of weeks since 0000-02-29.

Given two dates (Syntax 2), the WEEKS function returns the number of weeks between them. The
WEEKS function is similar to the DATEDIFF function, however the method used to calculate the number
of weeks between two dates is not the same and can return a different result. The return value for WEEKS
is determined by dividing the number of days between the two dates by seven, and then rounding down.
However, DATEDIFF uses number of week boundaries in its computation. This can cause the values
returned from the two functions to be different. For example, if the first date is a Friday and the second
date is the following Monday, the WEEKS function returns a difference of 0, but the DATEDIFF function
returns a difference of 1. While neither method is better than the other, you should consider the difference
when choosing between WEEKS and DATEDIFF.

For more information about the DATEDIFF function, see “DATEDIFF function [Date and
time]” on page 205.

Given a date and an integer (Syntax 3), the WEEKS function adds the integer number of weeks to
timestamp-expression. With Syntax 3, you must explicitly cast timestamp-expression as a TIME, DATE,
or TIMESTAMP data type. If timestamp-expression is a TIME value, the current date is assumed. Instead
of Syntax 3, use the DATEADD function.

For more information about the DATEADD function, see “DATEADD function [Date and
time]” on page 204.

See also

For information about casting data types, see “CAST function [Data type conversion]” on page 174.

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns the value 8, signifying that 2008-09-13 10:07:12 is eight weeks after
2008-07-13 06:07:12.

        SELECT WEEKS( '2008-07-13 06:07:12', '2008-09-13 10:07:12' );

The following statement returns the value 104792, signifying that the date is 104792 weeks after
0000-02-29.

        SELECT WEEKS( '2008-07-13 06:07:12' );

The following statement returns the TIMESTAMP value 2008-06-16 21:05:07.0, indicating the date and
time five weeks after 2008-05-12 21:05:07.

        SELECT WEEKS( CAST( '2008-05-12 21:05:07' AS TIMESTAMP ), 5);

WRITE_CLIENT_FILE function [String]

Creates and writes to a file on the client computer.
Syntax

WRITE_CLIENT_FILE( filename, blob-expression [, 'A' ] )

Parameters

- **filename** The name of the file on the client computer. The name is resolved on the client computer relative to the current working directory of the client application.

- **blob-expression** A binary string to be written to filename on the client computer.

- **A** By default, if the file already exists, it is overwritten. If you want the data to be appended to existing data, specify 'A'. If the file does not already exist, and you specify 'A', the file is still created.

Returns

INT

Remarks

The database server converts filename from the database character set to the client character set. On the client computer, filename is then converted to the operating system character set.

Since the data is a binary string, if you want the data to be in a particular character set, or compressed, or encrypted, you must perform these operations on the data before sending it to the WRITE_CLIENT_FILE function.

Reading of the file is performed by the client software library and the transfer of data is done using the command sequence communication protocol.

Privileges

When writing to a file on a client computer:

- You must have the WRITE FILE system privilege.

- The client application must have write permissions on the computer being written to.

- The allow_write_client_file database option must be enabled.

- The write_client_file secure feature must be enabled.

See also

- “allow_write_client_file option” [SQL Anywhere Server - Database Administration]
- “-sf database server option” [SQL Anywhere Server - Database Administration]
- “Access to data on client computers” [SQL Anywhere Server - SQL Usage]
- “UNLOAD statement” on page 1027
- “CSCONVERT function [String]” on page 198
- “DECOMPRESS function [String]” on page 219
- “DECRYPT function [String]” on page 220
Standards and compatibility

- SQL/2008  Vendor extension.

**XMLAGG function [Aggregate]**

Generates a forest of XML elements from a collection of XML values.

**Syntax**

```
XMLAGG( expression [ ORDER BY order-by-expression ] )
```

**Parameters**

- **expression**  An XML value. The content is escaped unless the data type is XML. The order-by-expression orders the elements returned by the function.

- **order-by-expression**  An expression used to order the XML elements according to the value of this expression.

When an ORDER BY clause contains constants, they are interpreted by the optimizer and then replaced by an equivalent ORDER BY clause. For example, the optimizer interprets ORDER BY 'a' as ORDER BY expression.

A query block containing more than one aggregate function with valid ORDER BY clauses can be executed if the ORDER BY clauses can be logically combined into a single ORDER BY clause. For example, the following clauses:

```
ORDER BY expression1, 'a', expression2
ORDER BY expression1, 'b', expression2, 'c', expression3
```

are subsumed by the clause:

```
ORDER BY expression1, expression2, expression3
```

**Returns**

XML

**Remarks**

Any values that are NULL are omitted from the result. If all inputs are NULL, or there are no rows, the result is NULL. If you require a well-formed XML document, you must ensure that your query is written so that the generated XML has a single root element.

Data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in base64-encoded format when you execute a query that contains XMLAGG.

For an example of a query that uses the XMLAGG function with an ORDER BY clause, see “Use of the XMLAGG function” [SQL Anywhere Server - SQL Usage].
See also

- “Use of the XMLAGG function” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 XMLAGG is part of optional SQL/2008 language feature X034. The optional ORDER BY clause for the XMLAGG function comprises optional SQL/2008 language feature X035.

Example

The following statement generates an XML document that shows the orders placed by each customer.

```sql
SELECT XMLELEMENT( NAME "order",
    XMLATTRIBUTES( ID AS order_id ),
    ( SELECT XMLAGG(
        XMLELEMENT(
            NAME "Products",
            XMLATTRIBUTES( ProductID, Quantity AS "quantity_shipped" )
        )
    FROM SalesOrderItems soi
    WHERE soi.ID = so.ID
    ) )
AS products_ordered
FROM GROUPO.SalesOrders so
ORDER BY so.ID;
```

**XMLCONCAT function [String]**

Produces a forest of XML elements.

**Syntax**

```sql
XMLCONCAT( xml-value [, ... ]
```

**Parameters**

- `xml-value` The XML values to be concatenated.

**Returns**

XML

**Remarks**

Generates a forest of XML elements. In an unparsed XML document, a forest refers to the multiple root nodes within the document. NULL values are omitted from the result. If all the values are NULL, then NULL is returned. The XMLCONCAT function does not check whether the argument has a prolog. If you require a well-formed XML document, you must ensure that your query is written so that a single root element is generated.

Element content is always escaped unless the data type is XML. Data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in base64-encoded format when you execute a query that contains a XMLCONCAT function.
See also

- “Use of the XMLCONCAT function” [SQL Anywhere Server - SQL Usage]
- “XMLELEMENT function [String]” on page 412
- “XMLFOREST function [String]” on page 414
- “String functions” on page 152

Standards and compatibility

- SQL/2008  XMLCONCAT comprises part of the optional SQL/2008 language feature X020.

Example

The following query generates <CustomerID>, <cust.fname>, and <cust.lname> elements for each customer.

```sql
SELECT XMLCONCAT( XMLELEMENT ( NAME CustomerID, ID ),
    XMLELEMENT( NAME cust_fname, GivenName ),
    XMLELEMENT( NAME cust_lname, Surname )
) AS "Customer Information"
FROM GROUPO.CUSTOMERS
WHERE ID < 120;
```

XMLELEMENT function [String]

Produces an XML element within a query.

Syntax

```sql
XMLELEMENT( ( { NAME element-name-expression } | string-expression
    [ , XMLATTRIBUTES ( attribute-value-expression [ AS attribute-name ],... ) ]
    [ , element-content-expression,... ]
)
```

Parameters

- **element-name-expression**  An identifier. For each row, an XML element with the same name as the identifier is generated.

- **attribute-value-expression**  An attribute of the element. This optional argument allows you to specify an attribute value for the generated element. This argument specifies the attribute name and content. If the **attribute-value-expression** is a column name, then the attribute name defaults to the column name. You can change the attribute name by specifying the **attribute-name** argument.

- **element-content-expression**  The content of the element. This can be any string expression. You can specify an unlimited number of **element-content-expression** arguments and they are concatenated together. For example, the following SELECT statement returns the value <x>abcdef</x>:

```sql
SELECT XMLELEMENT( NAME x, 'abc', 'def' );
```

Returns

XML
Remarks
NULL element values and NULL attribute values are omitted from the result. The letter case for both
element and attribute names is taken from the query.

Element content is always escaped unless the data type is XML. Invalid element and attribute names are
also quoted. For example, consider the following statement:

```sql
SELECT XMLELEMENT('H1', f_get_page_heading() );
```

If the function `f_get_page_heading` is defined as RETURNS LONG VARCHAR or RETURNS
VARCHAR(1000), then the result is HTML encoded:

```sql
CREATE FUNCTION f_get_page_heading() RETURNS LONG VARCHAR
BEGIN
    RETURN ('<B>My Heading</B>');
END;
```

The above SELECT statement returns the following:

```html
<H1>&lt;B&gt;My Heading&lt;/B&gt;</H1>
```

If the function is declared as RETURNS XML, then the above SELECT statement returns the following:

```xml
<H1><B>My Heading</B></H1>
```

For more information about quoting and the XMLELEMENT function, see “Invalid names and SQL/
XML” [SQL Anywhere Server - SQL Usage].

XMLELEMENT functions can be nested to create a hierarchy. To return different elements at the same
level of the document hierarchy, use the XMLFOREST function.

For more information, see “XMLFOREST function [String]” on page 414.

Data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in
base64-encoded format when you execute a query that contains the XMLELEMENT function.

See also
- “Use of the XMLELEMENT function” [SQL Anywhere Server - SQL Usage]
- “XMLCONCAT function [String]” on page 411
- “XMLFOREST function [String]” on page 414
- “String functions” on page 152

Standards and compatibility
- SQL/2008  XMLELEMENT constitutes part of optional SQL/2008 language feature X031.
  Omitting the NAME keyword and using a string expression as the first argument is a vendor
  extension. SQL Anywhere does not support the optional OPTION clause with the XMLELEMENT
  function.

Example
The following example produces an <item_name> element for each product in the result set, where the
product name is the content of the element.
SELECT ID, XMLELEMENT( NAME item_name, p.Name )
FROM Products p
WHERE ID > 400;

The following example returns `<A HREF="http://www.ianywhere.com/
TARGET="_top">iAnywhere web site</A>`:

```
SELECT XMLELEMENT(
   'A',
   XMLATTRIBUTES( 'http://www.ianywhere.com/'
   AS "HREF", '_top' AS "TARGET"),
   'iAnywhere web site'
);
```

The following example returns `<table><tbody><tr align="center" valign="top"><td>Cell 1 info</td><td>Cell 2 info</td></tr></tbody></table>`:

```
SELECT XMLELEMENT( name "table",
   XMLELEMENT( name "tbody",
      XMLELEMENT( name "tr",
         XMLATTRIBUTES('center' AS "align", 'top' AS "valign"),
         XMLELEMENT( name "td", 'Cell 1 info' ),
         XMLELEMENT( name "td", 'Cell 2 info' )
      )
   );
```

The following example returns `<x>abcdef</x>','<custom_element>abcdef</custom_element>':

```
CREATE VARIABLE @my_element_name VARCHAR(200);
SET @my_element_name = 'custom_element';
SELECT XMLELEMENT( NAME x, 'abc', 'def' ),
   XMLELEMENT( @my_element_name,'abc', 'def' );
```

**XMLFOREST function [String]**

Generates a forest of XML elements.

**Syntax**

```
XMLFOREST( element-content-expression [ AS element-name ],... )
```

**Parameters**

- **element-content-expression**  A string. An element is generated for each element-content-expression argument that is specified. The element-content-expression value becomes the content of the element. For example, if you specify the EmployeeID column from the Employees table for this argument, then an <EmployeeID> element containing an EmployeeID value is generated for each value in the table.

Specify the element-name argument to assign a name other than the element-content-expression to the element, otherwise the element name defaults to the element-content-expression name.
Returns
XML

Remarks
Produces a forest of XML elements. In the unparsed XML document, a forest refers to the multiple root nodes within the document. When all the arguments to the XMLFOREST function are NULL, a NULL value is returned. If only some values are NULL, the NULL values are omitted from the result. Element content is always quoted unless the data type is XML. You cannot specify attributes using the XMLFOREST function. Use the XMLELEMENT function to specify attributes for generated elements.

For more information about the XMLELEMENT function, see “XMLELEMENT function [String]” on page 412.

Element names are escaped unless the data type is XML.

If you require a well-formed XML document, you must ensure that your query is written so that a single root element is generated.

Data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in base64-encoded format when you execute a query that contains XMLFOREST.

See also
● “Use of the XMLFOREST function” [SQL Anywhere Server - SQL Usage]
● “XMLELEMENT function [String]” on page 412
● “XMLCONCAT function [String]” on page 411
● “String functions” on page 152

Standards and compatibility
● SQL/2008 XMLFOREST constitutes part of optional SQL/2008 language feature X032. SQL Anywhere does not support the optional XMLNAMESPACES clause, or the OPTION clause, with the XMLFOREST function.

Example
The following statement produces an XML element for the first and last name of each employee.

```
SELECT EmployeeID,
       XMLFOREST( GivenName, Surname )
       AS "Employee Name"
FROM Employees;
```

XMLGEN function [String]
Generates an XML value based on an XQuery constructor.

Syntax
```
XMLGEN( xquery-constructor, content-expression [ AS variable-name ],... )
```
Parameters

- **xquery-constructor**  An XQuery constructor. The XQuery constructor is an item defined in the XQuery language. It gives a syntax for constructing XML elements based on XQuery expressions. The `xquery-constructor` argument must be a well-formed XML document with one or more variable references. A variable reference is enclosed in curly braces and must be prefixed with a $ and have no surrounding white space. For example:

```sql
SELECT XMLGEN( '<a>{$x}</a>', 1 AS x );
```

- **content-expression**  A variable. You can specify multiple `content-expression` arguments. The optional `variable-name` argument is used to name the variable. For example:

```sql
SELECT XMLGEN( '<emp EmployeeID="{$EmployeeID}"'><StartDate>{$x}</StartDate></emp>', EmployeeID, StartDate AS x )
FROM GROUPO.Employees;
```

Returns

XML

Remarks

Computed constructors as defined in the XQuery specification are not supported by the XMLGEN function.

When you execute a query that contains an XMLGEN function, data in BINARY, LONG BINARY, IMAGE, and VARBINARY columns is automatically returned in base64-encoded format.

Element content is always escaped unless the data type is XML. Illegal XML element and attribute names are also escaped.

For information about escaping and the XMLGEN function, see “Invalid names and SQL/XML” [SQL Anywhere Server - SQL Usage].

See also

- “Use of the XMLGEN function” [SQL Anywhere Server - SQL Usage]
- “String functions” on page 152

Standards and compatibility

- **SQL/2008**  Vendor extension. XMLGEN provides similar functionality to the SQL/2008 XMLDOCUMENT function.

Example

The following example generates <emp>, <Surname>, <GivenName>, and <StartDate> elements for each employee.

```sql
SELECT XMLGEN( '<emp EmployeeID=">{$EmployeeID}">
    <Surname>{$Surname}</Surname>
    <GivenName>{$GivenName}</GivenName>
    <StartDate>{$StartDate}</StartDate>
  </emp>',
```

416 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
EmployeeID,
Surname,
GivenName,
StartDate
) AS employee_list
FROM GROUPO.Employees;

YEAR function [Date and time]

Returns the year component of the TIMESTAMP argument.

Syntax

YEAR( timestamp-expression )

Parameters

● timestamp-expression  A TIMESTAMP value.

Returns

SMALLINT

Remarks

The value returned is the years component of the given TIMESTAMP value, returned as a SMALLINT.

Standards and compatibility

● SQL/2008  Vendor extension.

Example

The following example returns the value 2001.

    SELECT YEAR( '2001-09-12' );

YEARS function [Date and time]

Manipulates a TIMESTAMP or returns the number of years between two TIMESTAMP values. See the Remarks section below.

Syntax 1

YEARS( timestamp-expression )

Syntax 2

YEARS( timestamp-expression, timestamp-expression )

Syntax 3

YEARS( timestamp-expression, integer-expression )
Parameters

- **timestamp-expression**  A date and time value of type TIMESTAMP.

- **integer-expression**  The number of years (as a SMALLINT value) to be added to timestamp-expression. If integer-expression is negative, the appropriate number of years is subtracted from timestamp-expression. If you supply an integer-expression, the timestamp-expression must be explicitly cast as a DATE, TIME, or TIMESTAMP value. If timestamp-expression is a TIME, the current year is assumed.

  For information about casting data types, see “CAST function [Data type conversion]” on page 174.

Returns

SMALLINT with Syntax 1 or Syntax 2.

TIMESTAMP with Syntax 3.

Remarks

The value of YEARS is computed by counting the number of first days of the year between the two dates.

See also

- “DATEDIFF function [Date and time]” on page 205
- “DATEADD function [Date and time]” on page 204

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statements both return -4.

```
SELECT YEARS( '1998-07-13 06:07:12',
              '1994-03-13 08:07:13' );

SELECT DATEDIFF( year,
                 '1998-07-13 06:07:12',
                 '1994-03-13 08:07:13' );
```

The following statements return 1998.

```
SELECT YEARS( '1998-07-13 06:07:12' )
SELECT DATEPART( year, '1998-07-13 06:07:12' );
```

The following statements return the given date advanced 300 years.

```
SELECT YEARS( CAST( '1998-07-13 06:07:12' AS TIMESTAMP ), 300 )
SELECT DATEADD( year, 300, '1998-07-13 06:07:12' );
```
YMD function [Date and time]

Returns a date value corresponding to the given year, month, and day of the month. Arguments are SMALLINT values from -32768 to 32767.

Syntax

YMD( smallint-expression1, smallint-expression2, smallint-expression3 )

Parameters

- **smallint-expression1**  The year.
- **smallint-expression2**  The number of the month. The year is adjusted if the month is outside the range 1-12.
- **smallint-expression3**  The day number. The day can be any integer; the date is adjusted accordingly.

Returns

DATE

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statement returns the value 1998-06-12.

```sql
SELECT YMD( 1998, 06, 12 );
```

If the values are outside their normal range, the date is adjusted accordingly. For example, the following statement returns the DATE value 2000-03-01.

```sql
SELECT YMD( 1999, 15, 1 );
```
SQL statements

This section describes the conventions used in the SQL statement documentation.

Common elements in SQL syntax

This section lists language elements that are found in the syntax of many SQL statements.

- **column-name**  An identifier that represents the name of a column.
- **condition**  An expression that evaluates to TRUE, FALSE, or UNKNOWN.
- **connection-name**  A string representing the name of an active connection.
- **data-type**  A storage data type.
- **expression**  An expression. A common example of an expression in syntax is a column name.
- **filename**  A string containing a file name.
- **hostvar**  A C language variable, declared as a host variable preceded by a colon.
- **materialized-view-name**  An identifier that represents the name of a materialized view.
- **number**  Any sequence of digits followed by an optional decimal part and preceded by an optional negative sign. Optionally, the number can be followed by an E and then an exponent. For example:
  
  42
  -4.038
  .001
  3.4e10
  1e-10

- **owner**  An identifier representing the user ID who owns a database object.
- **query-block**  A query block is a simple query expression, or a query expression with an ORDER BY clause.
- **query-expression**  A query expression can be a SELECT, UNION, INTERSECT, or EXCEPT block (that is, a statement that does not contain an ORDER BY, WITH, FOR, FOR XML, or OPTION clause), or any combination of such blocks.
- **role-name**  An identifier representing the role name of a foreign key. In conceptual database modeling, a verb or phrase that describes a relationship from one point of view. You can describe each relationship with two roles. Examples of roles are "contains" and "is a member of."
- **savepoint-name**  An identifier that represents the name of a savepoint.
- **search-condition**  A condition that evaluates to TRUE, FALSE, or UNKNOWN.
SQL statements

- **special-value** A special value.
- **statement-label** An identifier that represents the label of a loop or compound statement.
- **statement-list** A list of SQL statements, each ending with a semicolon.
- **string-expression** An expression that resolves to a string.
- **table-list** A list of table names, which may include correlation names.
- **table-name** An identifier that represents the name of a table.
- **userid** An identifier representing a user name.
- **variable-name** An identifier that represents a variable name.
- **window-name** An identifier that represents a window name. Used in syntax related to window definition (for example, the WINDOW clause, and window functions such as RANK).

**See also**
- “Savepoints within transactions” [SQL Anywhere Server - SQL Usage]
- “Variables” on page 79
- “Truth value search conditions” on page 63
- “Materialized views” [SQL Anywhere Server - SQL Usage]
- “Host variables in embedded SQL” [SQL Anywhere Server - Programming]
- “Expressions” on page 21
- “SQL data types” on page 89
- “Strings” on page 6
- “Database connections” [SQL Anywhere Server - Database Administration]
- “Control statements” [SQL Anywhere Server - SQL Usage]
- “Search conditions” on page 40
- “Special values” on page 65
- “FROM clause” on page 810
- “Key joins” [SQL Anywhere Server - SQL Usage]
- “Identifiers” on page 4

**Syntax conventions**

The following conventions are used in the SQL syntax descriptions:

- **Keywords** All SQL keywords appear in uppercase, like the SQL statement ALTER TABLE in the following example:

```
ALTER TABLE [ owner.]table-name
```

- **Placeholders** Items that must be replaced with appropriate identifiers or expressions appear in italics, like the words `owner` and `table-name` in the following example:

```
ALTER TABLE [ owner.]table-name
```
• **Clause order**  If the order of optional clauses is significant in SQL statement syntax, the clauses are listed in the main body of the syntax in the order in which they should be listed, similar to the following:

```
CREATE SYNCHRONIZATION SUBSCRIPTION [ subscription-name ]
TO publication-name
[ FOR ml-username, ... ]
```

In the case where the order of optional clauses is not significant in SQL statement syntax, the clauses are listed separately like a list of options, similar to the following:

```
CREATE [ OR REPLACE ] SPATIAL REFERENCE SYSTEM
srs-name
[ srs-attribute ] [ srs-attribute ... ]
```

- **Optional portions**  Optional portions of a statement are enclosed by square brackets. For example:

```
RELEASE SAVEPOINT [ savepoint-name ]
```

These square brackets indicate that the `savepoint-name` is optional. The square brackets should not be typed.

You might also see square brackets around a portions of keywords. For example, the following syntax indicates that you can use either COMMIT TRAN or COMMIT TRANSACTION:

```
COMMIT TRAN[SACTION] ...
```

Likewise, the following syntax indicates that you can use either COMMIT or COMMIT WORK:

```
COMMIT [ WORK ]
```

• **Repeating items**  An item that can be repeated is followed by the appropriate list separator and an ellipsis (three dots), like `column-constraint` in the following example:

```
ADD column-definition [ column-constraint, ... ]
```

In this case, you can specify no column constraint, one, or more. If more than one is specified, they must be separated by commas.

• **Options**  When none or only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in square brackets.

```
[ ASC | DESC ]
```

For example, you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

• **Alternatives**  When precisely one of the options must be chosen, the alternatives are enclosed in curly braces.
[ QUOTES { ON | OFF } ]

In this case, if the QUOTES option is chosen, one of ON or OFF must be provided. The brackets and braces should not be typed.

Statement applicability indicators

Some statement titles are followed by an indicator in square brackets that indicate where the statement can be used. These indicators are as follows:

- **[ESQL]** The statement is for use in embedded SQL.
- **[Interactive SQL]** The statement can be used only in Interactive SQL.
- **[SP]** The statement is for use in stored procedures, triggers, or batches.
- **[T-SQL]** The statement is implemented for compatibility with Adaptive Server Enterprise. Sometimes the statement cannot be used in stored procedures that are not in Transact-SQL format. In other cases, an alternative statement closer to the SQL/2008 standard is recommended unless Transact-SQL compatibility is an issue.
- **[External call]** The statement is for use in calling external functions and procedures.
- **[MobiLink]** The statement is for use only in MobiLink clients.
- **[SQL Remote]** The statement can be used only in SQL Remote.
- **[Web service]** The statement is for use in web services clients.

If two sets of brackets are used, the statement can be used in both environments. For example, [ESQL] [SP] means a statement can be used in both embedded SQL and stored procedures.

SQL statements

The following sections define the syntax information for all supported SQL statements.

**ALLOCATE DESCRIPTOR statement [ESQL]**

Allocates space for a SQL descriptor area (SQLDA).

**Syntax**

```sql
ALLOCATE DESCRIPTOR descriptor-name
[ WITH MAX { integer | hostvar } ]
```

descriptor-name : identifier
Parameters

**WITH MAX clause**  Allows you to specify the number of variables within the descriptor area. The default size is one. You must still call fill_sqlda to allocate space for the actual data items before doing a fetch or any statement that accesses the data within a descriptor area.

Remarks

Allocates space for a descriptor area (SQLDA). You must declare the following in your C code before using this statement:

```c
struct sqlda * descriptor_name
```

Privileges

None.

Side effects

None.

See also

- “DEALLOCATE DESCRIPTOR statement [ESQL]” on page 727
- “The SQL descriptor area (SQLDA)” [SQL Anywhere Server - Programming]

Standards and compatibility

- **SQL/2008**  ALLOCATE DESCRIPTOR is part of optional SQL language feature B031 "Basic dynamic SQL" of the SQL/2008 standard.

Example

The following sample program includes an example of ALLOCATE DESCRIPTOR statement usage.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA;
#include "sqldef.h"
EXEC SQL BEGIN DECLARE SECTION;
int         x;
short       type;
int         numcols;
char        string[100];
a_SQL_statement_number  stmt = 0;
EXEC SQL END DECLARE SECTION;
int main(int argc, char * argv[]){
    struct sqlda *      sqlda1;
    if( !db_init( &sqlca ) ) {
        return 1;
    }
    db_string_connect( &sqlca,
            "UID=DBA;PWD=sql;DBF=d:\DB Files\sample.db" );
    EXEC SQL ALLOCATE DESCRIPTOR sqlda1 WITH MAX 25;
    EXEC SQL PREPARE :stmt FROM
        'SELECT * FROM Employees';
    EXEC SQL DECLARE curs CURSOR FOR :stmt;
    EXEC SQL OPEN curs;
    EXEC SQL DESCRIBE :stmt into sqlda1;
```
EXEC SQL GET DESCRIPTOR sqlda1 :numcols=COUNT;
// how many columns?
if( numcols > 25 ) {
    // reallocate if necessary
    EXEC SQL DEALLOCATE DESCRIPTOR sqlda1;
    EXEC SQL ALLOCATE DESCRIPTOR sqlda1
        WITH MAX :numcols;
    EXEC SQL DESCRIBE :stmt into sqlda1;
} 

EXEC SQL GET DESCRIPTOR sqlda1 :numcols=COUNT;
// how many columns?
if( numcols > 25 ) {
    // reallocate if necessary
    EXEC SQL DEALLOCATE DESCRIPTOR sqlda1;
    EXEC SQL ALLOCATE DESCRIPTOR sqlda1
        WITH MAX :numcols;
    EXEC SQL DESCRIBE :stmt into sqlda1;
} 

type = DT_STRING; // change the type to string
EXEC SQL SET DESCRIPTOR sqlda1 VALUE 2 TYPE = :type;
fill_sqlda( sqlda1 );
// allocate space for the variables
EXEC SQL FETCH ABSOLUTE 1 curs
    USING DESCRIPTOR sqlda1;
EXEC SQL GET DESCRIPTOR sqlda1
    VALUE 2 :string = DATA;
printf("name = %s", string );
EXEC SQL DEALLOCATE DESCRIPTOR sqlda1;
EXEC SQL CLOSE curs;
EXEC SQL DROP STATEMENT :stmt;
db_string_disconnect( &sqlca, "" );
db_fini( &sqlca );
return 0;

ALTER DATABASE statement
Upgrades the database, turns jConnect support for a database on or off, calibrates the database, changes
the transaction log and transaction log mirror file names, or forces a mirror server to take ownership of a
database.

Syntax 1 - Upgrading components or restoring objects
ALTER DATABASE UPGRADE
    [PROCEDURE ON ]
    [JCONNECT { ON [ OFF ] }]
    [RESTART { ON [ OFF ] }]
    [SYSTEM PROCEDURE AS DEFINER { ON [ OFF ] }]

Syntax 2 - Performing calibration
ALTER DATABASE
    { CALIBRATE [ SERVER ]
        CALIBRATE DBSPACE dbspace-name
        CALIBRATE DBSPACE TEMPORARY
        CALIBRATE GROUP READ
        CALIBRATE PARALLEL READ
        RESTORE DEFAULT CALIBRATION
    }

Syntax 3 - Changing transaction log and transaction log mirror names
ALTER DATABASE dbfile
ALTER [ TRANSACTION ] LOG
    [ ON [ log-name ] [ MIRROR mirror-name ] ] OFF
    [ KEY key ]
Syntax 4 - Changing ownership of a database

ALTER DATABASE
dbname
  FORCE START
  SET PARTNER FAILOVER

Syntax 5 - Changing checksum settings

ALTER DATABASE dbfile
CHECKSUM OFF

Parameters

PROCEDURE clause  Drop and recreate all dbo- and SYS-owned procedures in the database.

JCONNECT clause  To allow the jConnect JDBC driver access to system catalog information, specify JCONNECT ON. This clause installs the system objects that provide jConnect support. Specify JCONNECT OFF to exclude the jConnect system objects. You can still use JDBC, as long as you do not access system information. JCONNECT is ON by default.

RESTART clause  RESTART is ON by default. When RESTART ON is specified and the AutoStop connection parameter is set to No, the database restarts after it is upgraded. Otherwise, the database is stopped after an upgrade.

SYSTEM PROCEDURE AS DEFINER { ON | OFF } clause  The SYSTEM PROCEDURE AS DEFINER clause specifies whether to execute pre-16.0 system procedures that perform privileged tasks with the privileges of the invoker or the definer (owner). ON means these system procedures are executed with the privileges of the definer (owner). OFF means these system procedures are executed with the privileges of the invoker.

If this clause is not specified, the default is to maintain the current behavior of the database being upgraded. When upgrading a pre-16.0 database, this means running the procedures as definer.

This setting does not impact user-defined procedures, or any procedures introduced in version 16.0 or later. For information about what system procedures this affects, and impacts of the setting, see “Running pre-16.0 system procedures as invoker or definer” [SQL Anywhere Server - SQL Usage].

CALIBRATE [ SERVER ] clause  Calibrate all dbspaces except for the temporary dbspace. This clause also performs the work done by CALIBRATE PARALLEL READ.

CALIBRATE DBSPACE clause  Calibrate the specified dbspace.

CALIBRATE DBSPACE TEMPORARY clause  Calibrate the temporary dbspace.

CALIBRATE GROUP READ clause  Perform group read calibration on the temporary dbspace. Writes large work tables to the temporary dbspace and uses different group read sizes to time the reading of the files. If adding space to the temporary table exceeds the limit for the connection, or if the cache is not large enough to allow calibration with the largest memory size, calibration fails and an error message is returned.

CALIBRATE PARALLEL READ clause  Calibrate the parallel I/O capabilities of devices for all dbspace files. The CALIBRATE [ SERVER ] clause also performs this calibration.
RESTORE DEFAULT CALIBRATION clause  Restore the Disk Transfer Time (DTT) model to the built-in default values that are based on typical hardware and configuration settings.

ALTER [ TRANSACTION ] LOG clause  Change the file name of the transaction log or transaction log mirror file. If MIRROR mirror-name is not specified, the clause sets a file name for a new transaction log. If the database is not currently using a transaction log, it starts using one. If the database is already using a transaction log, it starts using the new file as its transaction log.

If MIRROR mirror-name is specified, the clause sets a file name for a new transaction log mirror. If the database is not currently using a transaction log mirror, it starts using one. If the database is already using a transaction log mirror, it starts using the new file as its transaction log mirror.

You can also use this clause to turn off the transaction log or transaction log mirror. For example, ALTER DATABASE ALTER LOG OFF.

KEY clause  Specify the encryption key to use for the transaction log or transaction log mirror. The encryption key can be either a string or a variable name. When using the ALTER [ TRANSACTION ] LOG clause on a strongly encrypted database, you must specify the encryption key.

dbname FORCE START clause  Force a database server that is currently acting as the mirror server to take ownership of the database.

Caution
Using the FORCE START clause can result in the loss of transactions if the primary server contains transactions that the mirror server does not have.

It is recommended that you restart the primary and execute ALTER DATABASE with the SET PARTNER FAILOVER clause to force a failure without lost transactions. The FORCE START clause should only be used when the primary cannot be restarted as a last resort. See “Troubleshooting: The primary server cannot be restarted” [SQL Anywhere Server - Database Administration].

This clause can be executed from within a procedure or event, and must be executed while connected to the utility database on the mirror server.

SET PARTNER FAILOVER clause  Initiate a database mirroring failover from the primary server to the mirror server without stopping the server. This statement must be executed while you are connected to the database on the primary server, and can be executed from within a procedure or event. When this statement is executed:

1. The database server closes all connections to the database, including the connection that executed the statement
2. The database stops, and then restarts in the mirror role.
3. If the statement is contained in a procedure or event, then the other statements that follow it may not be executed

CHECKSUM clause  Disable global checksums for the database. By default, new databases have global checksums enabled, while version 11 and earlier databases do not have global checksums enabled.
Regardless of the setting of this clause, the database server always enables write checksums for databases running on storage devices such as removable drives, and databases running on Windows Mobile to help provide early detection if the database file becomes corrupt. The database server also calculates checksums for critical pages during validation activities.

For databases that do not have global checksums enabled, you can enable write checksums by using the -wc options.

Remarks

- **Syntax 1** Use the ALTER DATABASE UPGRADE statement as an alternative to the Upgrade utility (dbupgrad) to upgrade or update a database. By default, the database is stopped and restarted after the upgrade. The transaction log is archived during the upgrade and a new transaction log is created before the database is stopped or restarted.

In general, changes in databases between minor versions are limited to additional database options and minor system table and system procedure changes. The ALTER DATABASE UPGRADE statement upgrades the system tables to the current version and adds any new database options. If necessary, it also drops and recreates all system procedures. You can force a rebuild of the system procedures by specifying the PROCEDURE ON clause.

An error message is returned if you execute an ALTER DATABASE UPGRADE statement on a database that is currently being mirrored.

You can also use the ALTER DATABASE UPGRADE statement to restore settings and system objects to their original installed state.

Features that require a physical reorganization of the database file are not made available by executing an ALTER DATABASE UPGRADE statement. Such features include index enhancements and changes in data storage. To obtain the benefits of these enhancements, you must unload and reload your database.

---

**Caution**

Back up your database files before attempting to upgrade your database.

To use the jConnect JDBC driver to access system catalog information, specify JCONNECT ON (the default). To exclude the jConnect system objects, specify JCONNECT OFF. Setting JCONNECT OFF does not remove jConnect support from a database. You can still use JDBC, as long as you do not access system catalog information. If you subsequently download a more recent version of jConnect, you can upgrade the version in the database by (re-)executing the ALTER DATABASE UPGRADE JCONNECT ON statement.

- **Syntax 2** Use Syntax 2 to perform recalibration of the I/O cost model used by the optimizer. This operation updates the Disk Transfer Time (DTT) model, which is a mathematical model of the disk I/O used by the cost model. When you recalibrate the I/O cost model, the database server is unavailable for other use. In addition, it is essential that all other activities on the computer are idle. Recalibrating the database server is an expensive operation and may take some time to complete. It is recommended that you leave the default in place.
When using the CALIBRATE PARALLEL READ clause, parallel calibration is not performed on dbspace files with fewer than 10,000 pages. Even though the database server automatically suspends all of its activity during calibration operations, parallel calibration should be done when there are no processes consuming significant resources on the same computer. After calibration, you can retrieve the maximum estimated number of parallel I/O operations allowed on a dbspace file by using the IOParallelism extended database property.

To eliminate repetitive, time-consuming recalibration activities when there is a large number of similar hardware installations, you can re-use a calibration by unloading it and then applying it (loading it) into another database by using the sa_unload_cost_model and sa_load_cost_model system procedures, respectively.

- **Syntax 3** Use the ALTER DATABASE statement to change the transaction log and transaction log mirror names associated with a database file. The database must not be running to make these changes. These changes are the same as those made by the Transaction Log (dblog) utility. You can execute this statement while connected to the utility database or another database, depending on the setting of the -gu option.

  If you are changing the transaction log or transaction log mirror of an encrypted database, you must specify a key. You cannot stop using the transaction log if the database is using auditing. Once you turn off auditing, you can stop using the transaction log. This syntax is not supported in procedures, triggers, events, or batches.

  Use the BACKUP DATABASE statement to rename the transaction log for a running database. For example:

```sql
BACKUP DATABASE DIRECTORY 'directory-name'
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

- **Syntax 4** ALTER DATABASE...FORCE START must be executed from the mirror server, not the primary server.

- **Syntax 5** This clause can only be used to disable checksums for a database.

ALTER DATABASE UPGRADE is not supported on Windows Mobile.

**Privileges**

- Syntax 1 and 2: you must have the ALTER DATABASE system privilege, and must be the only connection to the database.

- Syntax 3: you must have the SERVER OPERATOR system privilege, you must have file permissions on the directories where the transaction log is located, and the database must not be running.

  Your ability to execute the ALTER DATABASE dbfile ALTER TRANSACTION LOG statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

- Syntax 4: you must have the SERVER OPERATOR system privilege.

  The privileges required to execute an ALTER DATABASE dbname FORCE START statement can be changed by the -gd database server option.
• Syntax 5: you must have the ALTER DATABASE system privilege.

Side effects

Automatic commit

Syntax 1: The transaction log is archived during the upgrade. A new transaction log is created when the database restarts after the upgrade.

Syntax 1: The database is stopped at the end of the upgrade and by default is restarted.

Standards and compatibility

• SQL/2008 Vendor extension.

• Transact-SQL The ALTER DATABASE statement is supported by Adaptive Server Enterprise. However, the statement's clauses supported by Adaptive Server Enterprise are disjoint from those clauses supported by SQL Anywhere.

Example

The following example disables jConnect support:

```
ALTER DATABASE UPGRADE JCONNECT OFF;
```

The following example sets the transaction log file name associated with demo.db to mynewdemo.log:

```
ALTER DATABASE 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.db'
    ALTER LOG ON 'mynewdemo.log';
```
ALTER DBSPACE statement

Preallocates space for a dbspace or for the transaction log, or updates the catalog when a dbspace file is renamed or moved.

Syntax

```
ALTER DBSPACE { dbspace-name | TRANSLOG | TEMPORARY }
{ ADD number [ add-unit ]
  | RENAME filename }
```

```add-unit : PAGES | KB | MB | GB | TB```

Parameters

**TRANSLOG clause** You supply the special dbspace name TRANSLOG to preallocate disk space for the transaction log. Preallocation improves performance if the transaction log is expected to grow quickly.
You may want to use this feature if, for example, you are handling many binary large objects (BLOBs) such as bitmaps.

The syntax `ALTER DBSPACE dbspace-name TRANSLOG RENAME filename` is not supported.

**TEMPORARY clause** You supply the special dbspace name TEMPORARY to add space to temporary dbspaces. When space is added to a temporary dbspace, the additional space materializes in the corresponding temporary file immediately. Preallocating space to the temporary dbspace of a database can improve performance during execution complex queries that use large work tables.

**ADD clause** An ALTER DBSPACE statement with the ADD clause preallocates disk space for a dbspace. It extends the corresponding database file by the specified size, in units of pages, kilobytes (KB), megabytes (MB), gigabytes (GB), or terabytes (TB). If you do not specify a unit, PAGES is the default. The page size of a database is fixed when the database is created.

If space is not preallocated, database files are extended by about 256 KB at a time for page sizes of 2 KB, 4 KB, and 8 KB, and by about 32 pages for other page sizes, when the space is needed. Pre-allocating space can improve performance for loading large amounts of data and also serves to keep the database files more contiguous within the file system.

You can use this clause to add space to any of the predefined dbspaces (system, temporary, temp, translog, and translogmirror).

**RENAME clause** If you rename or move a database file other than the main file to a different directory or device, you can use ALTER DBSPACE with the RENAME clause to ensure that SQL Anywhere finds the new file when the database is started. The `filename` parameter can be a string literal, or a variable.

*Cloud note:* For tenant databases in a cloud, when you specify the location of a dbspace, you can specify only a file name. You cannot specify a directory path.

The name change takes effect as follows:

- If the dbspace was already open before the statement was executed (that is, you have not yet renamed the actual file), it remains accessible; however, the name stored in the catalog is updated. After the database is stopped, you must rename the file to match what you provided using the RENAME clause, otherwise the file name won't match the dbspace name in the catalog and the database server is unable to open the dbspace the next time the database is started.

- If the dbspace was not open when the statement was executed, the database server attempts to open it after updating the catalog. If the dbspace can be opened, it becomes accessible. No error is returned if the dbspace cannot be opened.

To determine if a dbspace is open, execute the statement below. If the result is NULL, the dbspace is not open.

```sql
SELECT DB_EXTENDED_PROPERTY('FileSize', 'dbspace-name');
```

Using ALTER DBSPACE with RENAME on the main dbspace, system, has no effect. The RENAME clause is not supported for changing the name of the transaction log file. You can use the BACKUP DATABASE statement to rename the transaction log for a running database. For example:
BACKUP DATABASE DIRECTORY 'directory-name'
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;

Remarks
Each database is held in one or more files. A dbspace is an additional file with a logical name associated
with each database file, and used to hold more data than can be held in the main database file alone.
ALTER DBSPACE modifies the main dbspace (also called the root file) or an additional dbspace. The
dbspace names for a database are held in the SYSDBSPACE system view. The main database file has a
dbspace name of system.

When a multi-file database is started, the start line or ODBC data source description tells SQL Anywhere
where to find the main database file. The main database file holds the system tables. SQL Anywhere
looks in these system tables to find the location of the other dbspaces, and then opens each of the other
dbspaces. You can specify which dbspace new tables are created in by setting the default_dbspace option.

Privileges
You must have the MANAGE ANY DBSPACE system privilege.

Side effects
Automatic commit.

See also
- “CREATE DBSPACE statement” on page 557
- “BACKUP statement” on page 516
- “SYSDBSPACE system view” on page 1354
- “default_dbspace option” [SQL Anywhere Server - Database Administration]
- “Predefined dbspaces” [SQL Anywhere Server - Database Administration]
- “Database file types” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example increases the size of the system dbspace by 200 pages:

ALTER DBSPACE system
ADD 200;

The following example increases the size of the system dbspace by 400 MB:

ALTER DBSPACE system
ADD 400 MB;

The following example changes the file name associated with the fictitious system_2 dbspace:

ALTER DBSPACE system_2
RENAME 'e:\db\dbspace2.db';
ALTER DOMAIN statement

Renames a user-defined domain or data type.

Syntax

ALTER { DOMAIN | DATATYPE } user-type
RENAME new-name

Remarks

When you execute this statement, the name of the user-defined domain or data type is updated in the ISYSUSERTYPE system table.

Note

Any procedures, triggers, views, or events that refer to the user-defined domain or data type must be recreated, or else they continue to refer to the old name.

Privileges

You must be the owner of the domain, or have one of the following privileges:

- ALTER privilege on the domain
- ALTER DATATYPE system privilege
- ALTER ANY OBJECT system privilege

Side effects

Automatic commit.

See also

- “SYSUSERTYPE system view” on page 1415
- “CREATE DOMAIN statement” on page 562
- “Domains” on page 130
- “Domains” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008  Vendor extension. The ALTER DOMAIN statement is optional SQL feature F711 of the SQL/2008 standard. However, in the standard, ALTER DOMAIN can specify modified DEFAULT or CHECK constraint clauses for an existing domain. Neither of these operations are supported in SQL Anywhere. Feature F711 does not support the renaming of a domain.

Example

The following example renames the fictitious Address domain to MailingAddress:

ALTER DOMAIN Address RENAME MailingAddress;
ALTER EVENT statement

Changes the definition of an event or its associated handler for automating predefined actions, or alters
the definition of scheduled actions. You can also use this statement to hide the definition of an event
handler.

Syntax 1 - Altering an event

ALTER EVENT [ owner.]event-name
[ AT { CONSOLIDATED | REMOTE | ALL } ]
[ FOR { PRIMARY | ALL } ]
[ { DELETE TYPE
  | TYPE event-type
  | WHERE { trigger-condition | NULL } 
  | { ADD | ALTER | DELETE } SCHEDULE schedule-spec } ]
[ ENABLE | DISABLE ]
[ [ ALTER ] HANDLER compound-statement | DELETE HANDLER ]

event-type :
  BackupEnd
  Connect
  ConnectFailed
  DatabaseStart
  DBDiskSpace
  Deadlock
  "Disconnect"
  GlobalAutoincrement
  GrowDB
  GrowLog
  GrowTemp
  LogDiskSpace
  RAISERROR
  ServerIdle
  TempDiskSpace

trigger-condition :
event_condition( condition-name ) { = | < | > | != | <= | >= } value

schedule-spec :
[ schedule-name ]
[ { START TIME start-time | BETWEEN start-time AND end-time } ]
[ EVERY period { HOURS | MINUTES | SECONDS } ]
[ ON { ( day-of-week, ... ) | ( day-of-month, ... ) } ]
[ START DATE start-date ]

event-name | schedule-name : identifier

day-of-week : string

value | period | day-of-month : integer

start-time | end-time : time

start-date : date
Syntax 2 - Hiding the definition of an event handler

```
ALTER EVENT event-name SET HIDDEN
```

Parameters

- **AT clause**  Use this clause to change the specification regarding the databases at which the event is handled.

- **FOR clause**  Use this clause in a database mirroring or read-only scale-out system to restrict the databases at which the event is handled.

- **DELETE TYPE clause**  Use this clause to remove an association of the event with an event type.

- **ADD | ALTER | DELETE SCHEDULE clause**  Use this clause to change the definition of a schedule. Only one schedule can be altered in any one ALTER EVENT statement.

- **WHERE clause**  Use this clause to change the trigger condition under which an event is fired. The WHERE NULL option deletes a condition.

- **START TIME clause**  Use this clause to specify the start time and, optionally, the end time, for the event. The `start-time` and `end-time` parameters are strings (for example, '12:34:56'). Variables and expressions are not allowed (for example, `NOW()`).

- **START DATE clause**  Use this clause to specify the start date for the event. The `start-date` parameter is a string. Variables and expressions are not allowed (for example, `TODAY()`).

- **SET HIDDEN clause**  Use this clause to hide the definition of an event handler. Specifying the SET HIDDEN clause results in the permanent obfuscation of the event handler definition stored in the action column of the ISYSEVENT system table.

Remarks

This statement allows you to alter an event definition created with CREATE EVENT. Possible uses include the following:

- hiding the definition of an event handler

- defining and testing an event handler without a trigger condition or schedule during a development phase, and then adding the conditions for execution using ALTER EVENT once the event handler is completed

If you need to alter an event, you can disable it while it is running by executing an ALTER EVENT...DISABLE statement. To disable an event in Sybase Central, right-click the event and clear the Enabled option. Disabling the event does not interrupt current event handler execution; the event handler continues to execute until completion. When the event handler completes, it is not restarted until you re-enable it. You can alter and then re-enable the definition. To determine what events are running, execute the following statement:

```
SELECT *
FROM dbo.sa_conn_info()
WHERE CONNECTIONPROPERTY( 'EventName', Number ) = 'event-name';
```
Privileges
You must have either the ALTER ANY OBJECT or MANAGE ANY EVENT system privilege.

Side effects
Automatic commit.

Standards and compatibility
- SQL/2008 Vendor extension.

See also
- “SYSEVENT system view” on page 1356
- “BEGIN statement” on page 523
- “CREATE EVENT statement” on page 570
- “System events” [SQL Anywhere Server - Database Administration]
- “Hiding an event handler” [SQL Anywhere Server - Database Administration]

ALTER EXTERNAL ENVIRONMENT statement
Specifies the location of an external environment such as Java, PHP, or Perl.

Syntax
```
ALTER EXTERNAL ENVIRONMENT environment-name
LOCATION location-string
```

Parameters
- `environment-name` Use `environment-name` to specify the external environment you are altering.

- **LOCATION clause** Use the LOCATION clause to specify the location on the database server computer where the executable/binary for the external environment can be found. It includes the executable/binary name. This path can either be fully qualified or relative. If the path is relative, then the executable/binary must be in a location where the server can find it.

Remarks
The location of dbmlsync utility can be set using this statement. If, during synchronization, the dbmlsync executable image cannot be located using the database server's PATH environment variable, then use this statement to record the location of the executable.
Privileges
You must have the MANAGE ANY EXTERNAL ENVIRONMENT system privilege.

Side effects
None

See also
● “SQL Anywhere external environment support” [SQL Anywhere Server - Programming]
● “START EXTERNAL ENVIRONMENT statement” on page 997
● “STOP EXTERNAL ENVIRONMENT statement” on page 1005
● “INSTALL EXTERNAL OBJECT statement” on page 866
● “REMOVE EXTERNAL OBJECT statement” on page 933
● “SYSEXTERNENV system view” on page 1358
● “SYSEXTERNENVOBJECT system view” on page 1360

Standards and compatibility
● SQL/2008  Vendor extension.

Example
The following example specifies the location of the Perl executable for use when using Perl as an external environment.

```
ALTER EXTERNAL ENVIRONMENT PERL
LOCATION 'c:\Perl64\bin\perl.exe';
```

ALTER FUNCTION statement
Modifies a function.

Syntax 1 - Change the definition of a function
```
ALTER FUNCTION [ owner.]function-name function-definition
```

function-definition : CREATE FUNCTION syntax

Syntax 2 - Obfuscate a function definition
```
ALTER FUNCTION [ owner.]function-name
SET HIDDEN
```

Syntax 3 - Recompile a function
```
ALTER FUNCTION [ owner.]function-name
RECOMPILE
```

Remarks
You must include the entire new function in the ALTER FUNCTION statement.

● Syntax 1  The ALTER FUNCTION statement is identical in syntax to the CREATE FUNCTION statement except for the first word.
With ALTER FUNCTION, existing privileges on the function remain unmodified. Conversely, if you execute DROP FUNCTION followed by CREATE FUNCTION, execute privileges are reassigned.

- Syntax 2 Use SET HIDDEN to obfuscate the definition of the associated function and cause it to become unreadable. The function can be unloaded and reloaded into other databases.

If SET HIDDEN is used, debugging using the debugger does not show the function definition, nor is it available through procedure profiling.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>This setting is irreversible. It is strongly recommended that you retain the original function definition outside of the database.</td>
</tr>
</tbody>
</table>

- Syntax 3 Use the RECOMPILE syntax to recompile a user-defined SQL function. When you recompile a function, the definition stored in the catalog is re-parsed and the syntax is verified. The preserved source for a function is not changed by recompiling. When you recompile a function, the definitions obfuscated by the SET HIDDEN clause remain obfuscated and unreadable.

Privileges

You must be the owner of the function or have one of the following privileges:

- ALTER ANY PROCEDURE system privilege
- ALTER ANY OBJECT system privilege

To make a function external, you must have the CREATE EXTERNAL REFERENCE system privilege.

Side effects

Automatic commit.

See also

- “CREATE FUNCTION statement” on page 594
- “CREATE FUNCTION statement [External call]” on page 580
- “CREATE FUNCTION statement [Web service]” on page 586
- “ALTER PROCEDURE statement” on page 456
- “DROP FUNCTION statement” on page 758
- “Hiding the contents of a procedure, function, trigger, event, or view” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 Vendor extension. ALTER FUNCTION is optional SQL language feature F381 of the SQL/2008 standard. However, in the SQL standard, ALTER FUNCTION cannot be used to re-define a SQL Persistent Stored Module (PSM) function definition. SQL/2008 does not include support for SET HIDDEN or RECOMPILE.
Example

In this example, MyFunction is created and altered. The SET HIDDEN clause obfuscates the function definition and makes it unreadable. To run this example, you must also have the CREATE PROCEDURE system privilege, since a function is being created before being altered.

```
CREATE FUNCTION MyFunction(
    firstname CHAR(30),
    lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
    DECLARE name CHAR(61);
    SET name = firstname || ' ' || lastname;
    RETURN (name);
END;
```

```
ALTER FUNCTION MyFunction SET HIDDEN;
```

ALTER INDEX statement

Renames an index, primary key, or foreign key, or changes the clustered nature of an index.

Syntax

```
ALTER { INDEX index-name \\
    | [ INDEX ] FOREIGN KEY role-name \\
    | [ INDEX ] PRIMARY KEY } \\
ON [ owner.]object-name { REBUILD | rename-clause | cluster-clause }
```

- **object-name**: `table-name` | `materialized-view-name`
- **rename-clause**: `RENAME { AS | TO } new-index-name`
- **cluster-clause**: `CLUSTERED` | `NONCLUSTERED`

Parameters

- **rename-clause**: Specify the new name for the index, primary key, or foreign key.

When you rename the underlying index for a foreign or primary key, the corresponding RI constraint name for the index is not changed. However, the foreign key role name, if applicable, is the same as the index name and is changed. Use the ALTER TABLE statement to rename the RI constraint name, if necessary.

- **cluster-clause**: Specify whether the index should be changed to CLUSTERED or NONCLUSTERED. Only one index on a table can be clustered.

- **REBUILD clause**: Use this clause to rebuild an index, instead of dropping and recreating it.

Remarks

The ALTER INDEX statement carries out two tasks:

- It can be used to rename an index, primary key, or foreign key.
- It can be used to change an index type from nonclustered to clustered, or vice versa.
The ALTER INDEX statement can be used to change the clustering specification of the index, but does not reorganize the data. As well, only one index per table or materialized view can be clustered.

ALTER INDEX cannot be used to change an index on a local temporary table. An attempt to do so results in an Index not found error.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges
To alter an index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

To alter an index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

Side effects
Automatic commit. Clears the Results tab in the Results pane in Interactive SQL. Closes all cursors for the current connection. If ALTER INDEX REBUILD is specified, a checkpoint is performed.

See also
- “CREATE INDEX statement” on page 599
- “ALTER TABLE statement” on page 486
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement changes IX_product_name to be a clustered index:

```sql
ALTER INDEX IX_product_name ON GROUPO.Products
CLUSTERED;
```

The following statement renames the index IX_product_name on the Products table to ixProductName:

```sql
ALTER INDEX IX_product_name ON GROUPO.Products
RENAME TO ixProductName;
```

ALTER LDAP SERVER statement
Alters an LDAP server configuration object.
Syntax

```
ALTER LDAP SERVER ldapua-server-name
    [ ldapua-server-attribs ... ]
    [ WITH { SUSPEND | ACTIVATE | REFRESH } ]
```

```
ldapua-server-attribs :
    SEARCH DN search-dn-attributes ...
    | AUTHENTICATION URL { 'url-string' | NULL }
    | CONNECTION TIMEOUT timeout-value
    | CONNECTION RETRIES retry-value
    | TLS { ON | OFF }
```

```
search-dn-attributes :
    URL { 'url-string' | NULL }
    | ACCESS ACCOUNT { 'dn-string' | NULL }
    | IDENTIFIED BY { 'password' | NULL }
    | IDENTIFIED BY ENCRYPTED { encrypted-password | NULL }
```

Parameters

**SEARCH DN clause**  
There is no default value for any parameter in the SEARCH DN clause.

- **URL**  
Use this clause to specify the host (by name or by IP address), port number, and search to be performed to do the lookup of the **LDAP Distinguished Name (DN)** for a given user ID. `url-string` is validated for correct LDAP URL syntax before it is stored in ISYSLDAPSERVER. The maximum size for this string is 1024 bytes.


- **ACCESS ACCOUNT**  
Use this clause to specify the LDAP Distinguished Name (DN) used by the database server to connect to the LDAP server. This is not a SQL Anywhere user, but a user created in the LDAP server specifically for logging in to the LDAP server. This user must have permissions within the LDAP server to search for DNs by user ID in the locations specified in the SEARCH DN URL clause. The maximum size for this string is 1024 bytes.

- **IDENTIFIED BY**  
Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT. The maximum size is 255 bytes, and cannot be set to NULL.

- **IDENTIFIED BY ENCRYPTED**  
Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT, provided in encrypted form, and is a binary value stored somewhere on disk. The maximum size of the binary is 289 bytes, and cannot be set to NULL. IDENTIFIED BY ENCRYPTED allows the password to be retrieved and used, without it becoming known.

**AUTHENTICATION URL clause**  
Use this clause to specify the host by name or IP address, and the port number of the LDAP server to use to authenticate a user. The DN of the user obtained from a prior DN search and the user password are used to bind a new connection to the authentication URL. A successful connection to the LDAP server is considered proof of the identity of the connecting user. There is no default value for this parameter. For size limits to this string, see “SYSLDAPSERVER system view” on page 1369.
**CONNECTION TIMEOUT clause**  Use this clause to specify the connection timeout, in milliseconds, to the LDAP server, both for searches for the DN and for authentication. The default value is 10 seconds.

**CONNECTION RETRIES clause**  Use this clause to specify the number of retries for connections to the LDAP server, both for searches for the DN and for authentication. The valid range of values is 1-60. The default is 3.

**TLS clause**  Use this clause to specify the use of the TLS protocol on connections to the LDAP server, both for the DN searches, and for authentication. The valid values are ON or OFF. The default is OFF. Use the Secure LDAP protocol by specifying `ldaps://` to begin the URL instead of `ldap://`. The TLS option must be set to OFF when using Secure LDAP.

**WITH clause**

- **WITH SUSPEND**  Sets the state of the LDAP server communications to SUSPENDED (maintenance mode). The connections to the LDAP server are closed and authentication with the LDAP server is no longer performed.

- **WITH ACTIVATE**  Activates the LDAP server for immediate use. This changes the state of the LDAP server communications to READY.

- **WITH REFRESH**  Reinitializes LDAP user-authentication. This command does not change the state of the LDAP server if it is in the SUSPENDED state. When WITH REFRESH is specified for an LDAP server in the READY or ACTIVE state, connections to the LDAP server are closed. Then, the server option values are reread from the ISYSLDAPSERVER system table and are applied to new connections to the LDAP server and to incoming authentication requests to the database server.

**Remarks**

ALTER LDAP SERVER...WITH REFRESH is often used on an LDAP server that is in the ACTIVE or READY state to release any resources that may be held, or to reread changes made to files outside of the server, such as a change to the contents of the file specified by the trusted_certificates_file database option.

For other states, ALTER LDAP SERVER...WITH REFRESH has no effect.

**Privileges**

You must have the MANAGE ANY LDAP SERVER system privilege.

**Side effects**

Automatic commit.

**See also**

- “LDAP user authentication” [SQL Anywhere Server - SQL Usage]
- “CREATE LDAP SERVER statement” on page 603
- “DROP LDAP SERVER statement” on page 760
- “VALIDATE LDAP SERVER statement” on page 1044
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example suspends the fictitious LDAP server named apps_primary.

```
ALTER LDAP SERVER apps_primary WITH SUSPEND;
```

The following example changes the LDAP server named apps_primary to use a different URL for authentication on host fairfax, port number 1066, sets connection retries to 10, and activates it.

```
ALTER LDAP SERVER apps_primary
  AUTHENTICATION URL 'ldap://fairfax:1066/
  CONNECTION RETRIES 10
  WITH ACTIVATE;
```

ALTER LOGIN POLICY statement

Alters an existing login policy.

Syntax

```
ALTER LOGIN POLICY policy-name policy-options
```

```
policy options :
policy-option [ policy-option ... ]
```

```
policy-option :
policy-option-name = policy-option-value
```

```
policy-option-value :
{ UNLIMITED
  DEFAULT
  legal-option-value }
```

Parameters

- **policy-name**  The name of the login policy. Specify root to modify the root login policy.

- **policy-option-name**  The name of the policy option.

- **policy-option-value**  The value assigned to the login policy option. If you specify UNLIMITED, no limits are used. If you specify DEFAULT, the default limits are used.
<table>
<thead>
<tr>
<th>Policy-option-name</th>
<th>Description</th>
<th>Default value</th>
<th>Applies to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_unlock_time</td>
<td>The time period after which locked accounts are automatically unlocked.</td>
<td>Unlimited</td>
<td>All users except those with the MANAGE ANY USER system privilege</td>
</tr>
<tr>
<td>change_password_du-</td>
<td>When the value for this option is ON, setting the password requires two administrators.</td>
<td>OFF</td>
<td>All users</td>
</tr>
<tr>
<td>al_control</td>
<td>The setting for the verify_password_function option is ignored if this option is set to ON because the password is configured separately in two parts. No verification is performed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ldap_primary_server</td>
<td>The name of the primary LDAP server.</td>
<td>(none)</td>
<td>All users</td>
</tr>
<tr>
<td>ldap_secondary_server</td>
<td>The name of the secondary LDAP server.</td>
<td>(none)</td>
<td>All users</td>
</tr>
<tr>
<td>ldap_auto_failback_period</td>
<td>The time period in minutes after which automatic failback to primary server is attempted.</td>
<td>15 minutes</td>
<td>All users</td>
</tr>
<tr>
<td>ldap_failover_to_std</td>
<td>Whether to permit authentication with Standard authentication when authentication with the LDAP server fails due to failure to locate the Distinguished Name (DN) for a user, lack of system resources, network outage, connection timeouts, or similar system failures. This setting does not permit an actual authentication failure returned from an LDAP server to fail over to Standard authentication (as is the case when the user is located but the supplied password does not match).</td>
<td>ON</td>
<td>All users</td>
</tr>
<tr>
<td>Policy-option-name</td>
<td>Description</td>
<td>Default value</td>
<td>Applies to:</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ldap_refresh_dn</td>
<td>At the time this policy option is specified by a CREATE LOGIN POLICY or ALTER LOGIN POLICY statement, the current time value is stored with the login policy. This value is the timestamp against which user authentication compares the user_dn_cached_at value found for the user in ISYSUSER. If the value in the policy is newer than the user_dn_cached_at value in ISYSUSER, a search for a user's Distinguished Name (DN) is done to refresh the user_dn value in ISYSUSER. The value NOW is the only valid value to assign to this policy option. All others result in an error. The value is in Coordinated Universal Time (UTC) and is stored as a string in the server default format.</td>
<td>(none)</td>
<td>All users</td>
</tr>
<tr>
<td>locked</td>
<td>If the value for this option is ON, users are not allowed to establish new connections. The reason_locked column of the sa_get_user_status system procedure returns a string generated by the database server that shows why a user is locked.</td>
<td>OFF</td>
<td>All users except those with the MANAGE ANY USER system privilege</td>
</tr>
<tr>
<td>max_connections</td>
<td>The maximum number of concurrent connections allowed for a user.</td>
<td>Unlimited</td>
<td>All users except those with the SERVER OPERATOR or DROP CONNNECTION system privilege</td>
</tr>
<tr>
<td>max_failed_login_attempts</td>
<td>The maximum number of failed attempts since the last successful attempt to log in before the user is locked. Users with SYS_AUTH_DBA_ROLE compatibility role are unlocked after one minute has passed since the most recent failed login attempt.</td>
<td>Unlimited</td>
<td></td>
</tr>
</tbody>
</table>
### Policy-option-name | Description | Default value | Applies to:
--- | --- | --- | ---
max_days_since_login | The maximum number of days that can elapse between two successive logins by the same user. | Unlimited | All users except those with the MANAGE ANY USER system privilege
max_non_dba_connections | The maximum number of concurrent connections that users can make. This option is only supported in the root login policy. | Unlimited | All users except those with the SERVER OPERATOR or DROP CONNECTION system privilege
password_life_time | The maximum number of days before a password must be changed. | Unlimited | All users
password_grace_time | The number of days before the password expires during which login is allowed, but the default post_login procedure issues warnings. | 0 | All users
password_expiry_on_next_login | If the value for this option is ON, the user's password expires after the next login. | OFF | All users
root_auto_unlock_time | The time period after which locked accounts are automatically unlocked. This option is only supported in the root login policy. | 1 minute | Users with the MANAGE ANY USER system privilege

### Remarks
When a login policy is altered, changes are immediately applied to all users.

If you do not specify a policy option, values for this login policy are taken from the root login policy.
New policies do not inherit the MAX_NON_DBA_CONNECTIONS and ROOT_AUTO_UNLOCK_TIME policy options.
All new databases include a root login policy. You can modify the root login policy values, but you cannot delete the policy. An overview of the default values for the root login policy is provided in the table above.

**Privileges**

You must have the MANAGE ANY LOGIN POLICY system privilege.

**Side effects**

None.

**See also**

- “Altering a login policy” [SQL Anywhere Server - Database Administration]
- “ALTER USER statement” on page 508
- “COMMENT statement” on page 538
- “CREATE LOGIN POLICY statement” on page 607
- “CREATE USER statement” on page 721
- “DROP LOGIN POLICY statement” on page 761
- “DROP USER statement” on page 786
- “Login policies” [SQL Anywhere Server - Database Administration]

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Examples**

The following example alters the fictitious Test1 login policy by changing the LOCKED and MAX_CONNECTIONS policy options. The LOCKED value indicates that users with the policy cannot establish new connections and the MAX_CONNECTIONS value limits the number of concurrent connections that are allowed.

```
ALTER LOGIN POLICY Test1
  LOCKED=ON
  MAX_CONNECTIONS=5;
```

This example overrides the root login policy LOCKED and MAX_CONNECTIONS policy options.

```
ALTER LOGIN POLICY root
  LOCKED=ON
  MAX_CONNECTIONS=5;
```

The following example sets a primary and a secondary LDAP server for a fictitious ldap_user_policy login policy, and turns off the ability to failover to standard authentication, even when database option login_mode includes 'Standard'. This provides strict controls on users of this login policy so that only LDAP user authentication may be used for authentication. In the event that a high volume of login connections occur such that the LDAP server is unable to respond and authenticate quickly, users whose retries and timeouts are exhausted will see connection failures to the database server rather than failover to use standard authentication.

```
ALTER LOGIN POLICY ldap_user_policy
  LDAP_PRIMARY_SERVER=ldapsrv1
  LDAP_SECONDARY_SERVER=ldapsrv2
  LDAP_FAILOVER_TO_STD=OFF;
```
The following example resets the timestamp value for a fictitious application_user_policy login policy to
the current time. Any user that is assigned this policy have their Distinguished Name (DN) searched on
the next login attempt, rather than using the value cached in ISYSUSER. This strategy purges old DN
values held in ISYSUSER for users associated with this policy at the time of their next authentication.

```
ALTER LOGIN POLICY application_user_policy
LDAP_REFRESH_DN=NOW;
```

**ALTER MATERIALIZED VIEW statement**

Alters a materialized view.

**Syntax**

```
ALTER MATERIALIZED VIEW [ owner. ]materialized-view-name {  
  SET HIDDEN
  | { ENABLE | DISABLE } USE IN OPTIMIZATION
  | { ADD PCTFREE percent-free-space | DROP PCTFREE }
  | [ NOT ] ENCRYPTED
  | [ { IMMEDIATE | MANUAL } REFRESH ]
}
```

**Parameters**

- **SET HIDDEN clause**  Use the SET HIDDEN clause to obfuscate the definition of a materialized view.  
  *This setting is irreversible.*
- **ENABLE clause**  Use the ENABLE clause to enable a disabled materialized view, making it available
  for the database server to use. This clause has no effect on a view that is already enabled. After using this
  clause, you must refresh the view to initialize it, and recreate any text indexes that were dropped when the
  view was disabled.
- **DISABLE clause**  Use the DISABLE clause to disable use of the view by the database server. When
  you disable a materialized view, the database server drops the data and indexes for the view.
- **{ ENABLE | DISABLE } USE IN OPTIMIZATION clause**  Use this clause to specify whether you
  want the materialized view to be available for the optimizer to use. If you specify DISABLE USE IN
  OPTIMIZATION, the materialized view is used only when executing queries that explicitly reference the
  view. The default is ENABLE USE IN OPTIMIZATION.
- **ADD PCTFREE clause**  Specify the percentage of free space you want to reserve on each page. The
  free space is used if rows increase in size when the data is updated. If there is no free space on a page,
  every increase in the size of a row on that page requires the row to be split across multiple pages, causing
  row fragmentation and possible performance degradation.

The value of `percent-free-space` is an integer between 0 and 100. The value 0 specifies that no free space
is to be left on each page—each page is to be fully packed. A high value causes each row to be inserted
into a page by itself. If PCTFREE is not set, or is dropped, the default PCTFREE setting is applied
according to the database page size (200 bytes for a 4 KB page size, and 100 bytes for a 2 KB page size).
**DROP PCTFREE clause**  
Removes the PCTFREE setting currently in effect for the materialized view, and applies the default PCTFREE according to the database page size.

**[ NOT ] ENCRYPTED clause**  
Specify whether to encrypt the materialized view data. By default, materialized view data is not encrypted at creation time. To encrypt a materialized view, specify ENCRYPTED. To decrypt a materialized view, specify NOT ENCRYPTED.

**REFRESH clause**  
Use the REFRESH clause to change the refresh type for the materialized view:

- **IMMEDIATE REFRESH**  
  Use the IMMEDIATE REFRESH clause to change a manual view to an immediate view. The manual view must be valid and uninitialized to change the refresh type to IMMEDIATE REFRESH. If the view is in an initialized state, execute a TRUNCATE statement to change the state to uninitialized before executing the ALTER MATERIALIZED VIEW...IMMEDIATE REFRESH.

- **MANUAL REFRESH**  
  Use the MANUAL REFRESH clause to change an immediate view to a manual view.

**Remarks**

If you alter a materialized view owned by another user, you must qualify the name by including the owner (for example, GROUPO.EmployeeConfidential). If you don't qualify the name, the database server looks for a materialized view with that name owned by you and alters it. If there isn't one, it returns an error.

When you disable a materialized view (DISABLE clause), it is no longer available for the database server to use for answering queries. As well, the data and indexes are dropped, and the refresh type changes to manual. Any dependent regular views are also disabled.

The DISABLE clause requires exclusive access not only to the view being disabled, but to any dependent views, since they are also disabled.

Table encryption must already be enabled on the database to encrypt a materialized view (ENCRYPTED clause). The materialized view is then encrypted using the encryption key and algorithm specified at database creation time.

The only operations a user can perform on a materialized view to change its data are refreshing, truncating, and disabling. However, immediate views are automatically updated by the database server. That is, once an immediate view is enabled and initialized, the database server maintains it automatically, without additional privileges checking.

**Privileges**

You must be the owner of the materialized view, or have the ALTER ANY MATERIALIZED VIEW or ALTER ANY OBJECT system privilege.

If you do not have a required privilege but want to alter a materialized view to be immediate (ALTER MATERIALIZED VIEW...IMMEDIATE REFRESH), you must own the view and all the tables it references.
**Side effects**

- Automatic commit.

**See also**

- “CREATE MATERIALIZED VIEW statement” on page 612
- “REFRESH MATERIALIZED VIEW statement” on page 925
- “DROP MATERIALIZED VIEW statement” on page 762
- “TRUNCATE statement” on page 1019
- “sa_refresh_materialized_views system procedure” on page 1210
- “Materialized views” [SQL Anywhere Server - SQL Usage]
- “View dependencies” [SQL Anywhere Server - SQL Usage]
- “Restrictions when changing a materialized view from manual to immediate” [SQL Anywhere Server - SQL Usage]
- “Enabling or disabling a materialized view” [SQL Anywhere Server - SQL Usage]
- “Encrypting or decrypting a materialized view” [SQL Anywhere Server - SQL Usage]
- “Hiding a materialized view definition” [SQL Anywhere Server - SQL Usage]
- “Whether to set refresh type to manual or immediate” [SQL Anywhere Server - SQL Usage]
- “Advanced: Status and properties for materialized views” [SQL Anywhere Server - SQL Usage]
- “Enabling or disabling optimizer use of a materialized view” [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

The following statements creates the EmployeeConfid88 materialized view and then disables its use in optimization. To run this example you must also have the CREATE ANY MATERIALIZED VIEW system privilege, as well as SELECT privilege on the Employees and Departments tables.

```
CREATE MATERIALIZED VIEW EmployeeConfid88 AS
  SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary, ManagerID,
         Departments.DepartmentName, Departments.DepartmentHeadID
  FROM GROUP0.Employees, GROUP0.Departments
  WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid88;
ALTER MATERIALIZED VIEW EmployeeConfid88 DISABLE USE IN OPTIMIZATION;
```

**Caution**

When you are done with this example, you should drop the materialized view you created. Otherwise, you cannot make schema changes to its underlying tables, Employees and Departments, when trying out other examples. You cannot alter the schema of a table that has enabled, dependent materialized view.

```
CREATE MATERIALIZED VIEW EmployeeConfid88 AS
  SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary, ManagerID,
         Departments.DepartmentName, Departments.DepartmentHeadID
  FROM GROUP0.Employees, GROUP0.Departments
  WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid88;
ALTER MATERIALIZED VIEW EmployeeConfid88 DISABLE USE IN OPTIMIZATION;
```

**ALTER MIRROR SERVER statement**

**Note**

Read-only scale-out and database mirroring each require a separate license. See “Separately licensed components” [SQL Anywhere 16 - Introduction].
Modifies the attributes of a mirror server.

**Syntax 1**

```
ALTER MIRROR SERVER mirror-server-name
[AS { PRIMARY | MIRROR | ARBITER | PARTNER}]
[ server-option = { string | NULL } [ ... ] ]
```

**Syntax 2**

```
ALTER MIRROR SERVER mirror-server-name
[AS COPY]
[ { FROM SERVER parent-name [ OR SERVER server-name ] | USING AUTO PARENT } | ALTER PARENT FROM mirror-server-name ]
[ server-option = { string | NULL } [ ... ] ]
```

parent-name :
server-name | PRIMARY

server-option :
connection_string
logfile
preferred
state_file

**Parameters**

- **AS clause**  Use the AS clause to change the server-type of the mirror server from PARTNER to COPY or COPY to PARTNER. This clause is not needed and it is not recommended if you are not changing the type of mirror server.

  - **PARTNER**  Only a database server with the mirror server type of COPY can use this value to change its type to PARTNER. The parent definitions for the copy node are deleted.

    The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

    In a database mirroring system, the partners use the connection string value to connect to each other. In a read-only scale-out system, the connection string is used by a copy-node that has this server as its parent.

  - **COPY**  Only a database server whose mirror server type is PARTNER can use this value to change its server type to a copy node. This partner server must also currently have the MIRROR role.

    In a read-only scale-out system, this value specifies that the database server is a copy node. All connections to the database on this server are read-only. The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

- **FROM SERVER clause**  This clause can only be used when AS COPY is specified. This clause constructs a tree of servers for a scale-out system and indicates which servers the copy nodes obtain transaction log pages from.
The parent can be specified using the name of the mirror server or PRIMARY. An alternate parent for the copy node can be specified using the OR SERVER clause.

In a database mirroring system that has only two levels (partner and copy nodes), the copy nodes obtain transaction log pages from the current primary or mirror server.

A copy node determines which server to connect to by using its mirror server definition that is stored in the database. From its definition, it can locate the definition of its parent, and from its parent's definition, it can obtain the connection string to connect to the parent.

You do not have to explicitly define copy nodes for the scale-out system: you can choose to have the root node define the copy nodes when they connect.

**USING AUTO PARENT clause** This clause can only be used when AS COPY is specified. This clause causes the primary server to assign a parent for this server.

**ALTER PARENT FROM clause** This clause can only be used when AS COPY is specified. This clause changes the parent for this mirror server, and assigns all its siblings to be its children. The server name specified by the ALTER PARENT FROM clause is used to verify that the current parent for this server matches the value specified. This is used to ensure that only one of a collection of siblings is able to replace its parent if they all request the change simultaneously.

**server-option clause** The following options are supported:

○ **connection_string server option** Specifies the connection string to be used to connect to the server. The connection string for a mirror server should not include a user ID or password because they are not used when one mirror server connects to another mirror server.

  For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

○ **logfile server option** Specifies the location of the file that contains one line per request that is sent between mirror servers if database mirroring is used. This file is used only for debugging.

○ **preferred server option** Specifies whether the server is the preferred server in the mirroring system. You can specify either YES or NO. The preferred server assumes the role of primary server whenever possible. You specify this option when defining PARTNER servers.

○ **state_file server option** Specifies the location of the file used for maintaining state information about the mirroring system. This option is required for database mirroring. In a mirroring system, a state file must be specified for servers with type PARTNER. For arbiter servers, the location is specified as part of the command to start the server.

**Remarks**

In a database mirroring system, the mirror server type can be PRIMARY, MIRROR, ARBITER, or PARTNER.

In a read-only scale-out system, the mirror server type can be PRIMARY, PARTNER, or COPY.
You can only change the mirror server type from COPY to PARTNER or from PARTNER to COPY. To change to or from a PRIMARY, MIRROR, or ARBITER server type, you must drop the mirror server definition and recreate it.

Mirror server names for servers of type PARTNER and COPY must match the names of the database servers that will be part of the mirroring system (the name used with the -n server option). This requirement allows each database server to find its own definition and that of its parent. mirror-server-name, parent-name, and server-name above must contain only 7-bit ASCII characters.

When you convert from a copy node to a partner, the parent definitions are deleted from the mirror server definition.

To replace a mirror server definition, use the CREATE MIRROR SERVER statement with the OR REPLACE clause.

**Privileges**

You must have the MANAGE ANY MIRROR SERVER system privilege.

**Side effects**

Automatic commit.

**See also**

- “How child copy nodes are added” [SQL Anywhere Server - Database Administration]
- “Troubleshooting: State information files of the partners and arbiter” [SQL Anywhere Server - Database Administration]
- “Converting a partner server to a copy node” [SQL Anywhere Server - Database Administration]
- “Automatically assign the parent of a copy node” [SQL Anywhere Server - Database Administration]
- “Preferred database server in a database mirroring system” [SQL Anywhere Server - Database Administration]
- “SYSMIRRORSERVER system view” on page 1372
- “CREATE MIRROR SERVER statement” on page 615
- “COMMENT statement” on page 538
- “DROP MIRROR SERVER statement” on page 764

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

The following example does the following:

1. Creates a primary server called scaleout_primary1.
2. Creates a copy node, scaleout_child1, for scaleout_primary1.
3. Creates a mirror server, scaleout_mirror1.
4. Using the ALTER MIRROR SERVER statement, reassigns scaleout_child1 to be a copy node of scaleout_mirror1.
CREATE MIRROR SERVER "scaleout_primary1"
   AS PRIMARY
       connection_string = 'server=scaleout_primary1;host=winxp-2:6871,winxp-3:6872';
CREATE MIRROR SERVER "scaleout_child1"
   AS COPY FROM SERVER "scaleout_primary1"
       connection_string = 'server=scaleout_child1;host=winxp-2:6878';
CREATE MIRROR SERVER "scaleout_mirror1"
   AS MIRROR
       connection_string = 'server=scaleout_mirror1;host=winxp-2:6871,winxp-3:6872';
ALTER MIRROR SERVER "scaleout_child1"
   FROM SERVER "scaleout_mirror1"
       connection_string = 'server=scaleout_child1;host=winxp-2:6878';

ALTER PROCEDURE statement
Modifies a procedure.

Syntax 1
ALTER PROCEDURE [ owner.]procedure-name procedure-definition

procedure-definition : See "CREATE PROCEDURE statement" on page 639.

Syntax 2
ALTER PROCEDURE [ owner.]procedure-name
SET HIDDEN

Syntax 3
ALTER PROCEDURE [ owner.]procedure-name
RECOMPILE

Remarks
The ALTER PROCEDURE statement must include the entire new procedure. You can use PROC as a synonym for PROCEDURE.

● Syntax 1 The ALTER PROCEDURE statement is identical in syntax to the CREATE PROCEDURE statement except for the first word. Both Watcom and Transact-SQL dialect procedures can be altered through the use of ALTER PROCEDURE.

   With ALTER PROCEDURE, existing privileges on the procedure are not changed. If you execute DROP PROCEDURE followed by CREATE PROCEDURE, execute privileges are reassigned.

● Syntax 2 Use SET HIDDEN to obfuscate the definition of the associated procedure and cause it to become unreadable. The procedure can be unloaded and reloaded into other databases.

   If SET HIDDEN is used, debugging using the debugger does not show the procedure definition, and the definition is not available through procedure profiling.

   You cannot combine Syntax 2 with Syntax 1.
Note
This setting is irreversible. It is recommended that you retain the original procedure definition outside of the database.

● Syntax 3  Use the RECOMPILE syntax to recompile a stored procedure. When you recompile a procedure, the definition stored in the catalog is re-parsed and the syntax is verified. For procedures that generate a result set but do not include a RESULT clause, the database server attempts to determine the result set characteristics for the procedure and stores the information in the catalog. This can be useful if a table referenced by the procedure has been altered to add, remove, or rename columns since the procedure was created.

The procedure definition is not changed by recompiling. You can recompile procedures with definitions hidden with the SET HIDDEN clause, but their definitions remain hidden.

Privileges
You must be the owner of the procedure or have one of the following privileges:

● ALTER ANY PROCEDURE system privilege
● ALTER ANY OBJECT system privilege

Side effects
Automatic commit.

See also
● “CREATE PROCEDURE statement” on page 639
● “CREATE PROCEDURE statement [Web service]” on page 628
● “ALTER FUNCTION statement” on page 439
● “DROP PROCEDURE statement” on page 765
● “Hiding the contents of a procedure, function, trigger, event, or view” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
● SQL/2008  Vendor extension. ALTER PROCEDURE is optional SQL language feature F381 of the SQL/2008 standard. However, in the SQL standard, ALTER PROCEDURE cannot be used to re-define a stored procedure definition, and Transact-SQL dialect procedures are not supported. SQL/2008 does not include support for SET HIDDEN or RECOMPILE.

ALTERN PUBLICATION statement [MobiLink] [SQL Remote]
Alters a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, a publication identifies replicated data in both consolidated and remote databases.

Syntax
ALTER PUBLICATION [ owner.]publication-name alterpub-clause, ...
This statement is applicable only to MobiLink and SQL Remote.

The contribution to a publication from one table is called an article. Changes can be made to a publication by adding, modifying, or deleting articles, or by renaming the publication. If an article is modified, the entire definition of the modified article must be entered.

It is recommended that you perform a successful synchronization of a publication immediately before you alter it.

You cannot use the WHERE clause for publications that are defined as FOR DOWNLOAD ONLY or WITH SCRIPTED UPLOAD.

The SUBSCRIBE BY clause applies to SQL Remote only.

The USING clause is for scripted upload only.

You set options for a MobiLink publication with the ADD OPTION clause in the ALTER SYNCHRONIZATION SUBSCRIPTION statement or CREATE SYNCHRONIZATION SUBSCRIPTION statement.

When altering a MobiLink publication, an article can only be dropped after the execution of a START SYNCHRONIZATION SCHEMA CHANGE statement.

Requires exclusive access to all tables referred to in the statement as well as all tables in publication being modified.

Privileges

You must be the owner of the publication, or have one of the following privileges:

- ALTER privilege on the publication
- SYS_REPLICATION_ADMIN_ROLE system role

Side effects

Automatic commit.
### ALTER REMOTE MESSAGE TYPE statement [SQL Remote]

Changes the publisher's message system, or the publisher's address for a given message system, for a message type that has been created.

#### Syntax

```
ALTER REMOTE MESSAGE TYPE message-system
  ADDRESS address
```

- `message-system`: One of the message systems supported by SQL Remote. It must be one of the following values: `FILE`, `FTP`, or `SMTP`.
- `address`: A string containing a valid address for the specified message system.

#### Parameters

- `message-system`: One of the message systems supported by SQL Remote. It must be one of the following values: `FILE`, `FTP`, or `SMTP`.

- `address`: A string containing a valid address for the specified message system.

#### Remarks

The statement changes the publisher's address for a given message type.

The Message Agent sends outgoing messages from a database by one of the supported message links. The Extraction utility uses this address when it executes the GRANT CONSOLIDATE statement in the remote database.
The address is the publisher’s address under the specified message system. If it is an email system, the address string must be a valid email address. If it is a file-sharing system, the address string is a subdirectory of the directory specified by the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.

Privileges
You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects
Automatic commit.

See also
- “CREATE REMOTE [MESSAGE] TYPE statement [SQL Remote]” on page 651
- “GRANT CONSOLIDATE statement [SQL Remote]” on page 834
- “SQLREMOTE environment variable” [SQL Anywhere Server - Database Administration]
- “SQL Remote message systems” [SQL Remote]
- “Altering a message type” [SQL Remote]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement changes the publisher’s address for the FILE message link to new_addr.

```
ALTER REMOTE MESSAGE TYPE file
ADDRESS 'new_addr';
```

ALTER ROLE statement
Migrates a compatibility role to a user-defined role, and then drops the compatibility role.

Syntax
```
ALTER ROLE compatibility-role-name
MIGRATE TO new-role-name [, new-sa-role-name, new-sso-role-name ]
```

Parameters
- `compatibility-role-name` Use this parameter to specify the name of the compatibility role you are migrating.
- `new-role-name` Use this parameter to specify the name of the new role you are creating.
- `new-sa-role-name` Use this parameter to specify the name of the new role to migrate the SYS_AUTH_SA_ROLE role to. This parameter is required when migrating SYS_AUTH_DBA_ROLE, which causes SYS_AUTH_SA_ROLE to be migrated automatically.
new-sso-role-name  Use this parameter to specify the name of the new role to migrate the SYS_AUTH_SSO_ROLE role to. This parameter is required when migrating SYS_AUTH_DBA_ROLE, which causes SYS_AUTH_SSO_ROLE to be migrated automatically.

Remarks
The name of the new role must not begin and end with 'SYS_' and '_ROLE', respectively. For example SYS_MyBackup_ROLE is not an acceptable name for a user-defined role, whereas MyBackup_ROLE and SYS_MyBackup are acceptable.

When you execute the ALTER ROLE statement, grantees of the compatibility role are granted the new role.

You can restore migrated compatibility roles that have been migrated and then dropped by executing a CREATE ROLE statement and specifying the compatibility role name. For example, CREATE ROLE SYS_AUTH_BACKUP_ROLE; restores the SYS_AUTH_BACKUP_ROLE compatibility role.

Initially, only users with full administration rights (DBAs) can administer the new role, but you can use the CREATE ROLE statement with the OR REPLACE clause to specify additional administrators.

Use the GRANT or REVOKE statements to grant system privileges to the role, or revoke system privileges from the role.

You can migrate the SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE compatibility roles by migrating SYS_AUTH_DBA_ROLE compatibility, which causes SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE to be migrated automatically. When migrating SYS_AUTH_DBA_ROLE, you must include the new-sa-role-name and new-sso-role-name parameters to give new names to migrated SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE roles.

Privileges
You must have the MANAGE ROLES system privilege and administrative rights on the compatibility role you are migrating.

Side effects
None

See also
- “Roles” [SQL Anywhere Server - Database Administration]
- “CREATE ROLE statement” on page 652
- “min_role_admins option” [SQL Anywhere Server - Database Administration]
- “DROP ROLE statement” on page 769
- “REVOKE statement” on page 944

Standards and compatibility
- SQL/2008  Vendor extension.
Example

The following statement migrates all users and underlying system privileges granted to SYS_AUTH_BACKUP_ROLE role to a new role, custom_Backup_ROLE, and then drops SYS_AUTH_BACKUP_ROLE from the database.

```
ALTER ROLE SYS_AUTH_BACKUP_ROLE
MIGRATE TO custom_Backup_ROLE;
```

The following statement migrates all users, underlying system privileges, and roles granted to SYS_AUTH_DBA_ROLE compatibility role to a new role, custom_DBA. It then automatically migrates all users, underlying system privileges, and roles granted to SYS_AUTH_SA_ROLE and SYS_AUTH_SSO_ROLE to new roles called custom_SA and custom_SSO, respectively. Finally, it drops SYS_AUTH_DBA_ROLE, SYS_AUTH_SA_ROLE, and SYS_AUTH_SSO_ROLE from the database.

```
ALTER ROLE SYS_AUTH_DBA_ROLE
MIGRATE TO custom_DBA, custom_SA, custom_SSO;
```

ALTER SEQUENCE statement

Alters a sequence.

Syntax

```
ALTER SEQUENCE [ owner.] sequence-name
[ RESTART WITH signed-integer ]
[ INCREMENT BY signed-integer ]
[ MINVALUE signed-integer | NO MINVALUE ]
[ MAXVALUE signed-integer | NO MAXVALUE ]
[ CACHE integer | NO CACHE ]
[ CYCLE | NO CYCLE ]
```

Parameters

**RESTART WITH clause**  
Restarts the named sequence with the specified value.

**INCREMENT BY clause**  
Defines the amount the next sequence value is incremented from the last value assigned. The default is 1. Specify a negative value to generate a descending sequence. An error is returned if the INCREMENT BY value is 0.

**MINVALUE clause**  
Defines the smallest value generated by the sequence. The default is 1. An error is returned if MINVALUE is greater than \((2^{63}-1)\) or less than \(-(2^{63}-1)\). An error is also returned if MINVALUE is greater than MAXVALUE.

**MAXVALUE clause**  
Defines the largest value generated by the sequence. The default is \(2^{63}-1\). An error is returned if MAXVALUE is greater than \(2^{63}-1\) or less than \(-(2^{63}-1)\).

**CACHE clause**  
Specifies the number of preallocated sequence values that are kept in memory for faster access. When the cache is exhausted, the sequence cache is repopulated and a corresponding entry is written to the transaction log. At checkpoint time, the current value of the cache is forwarded to the ISYSSEQUENCE system table. The default is 100.
**CYCLE clause**  Specifies whether values should continue to be generated after the maximum or minimum value is reached.

**Remarks**
If the named sequence cannot be located, an error message is returned.

**Privileges**
You must be the owner of the sequence, or have one of the following privileges:

- ALTER ANY SEQUENCE system privilege
- ALTER ANY OBJECT system privilege

**Side effects**
None

**See also**
- “Use of a sequence to generate unique values” [SQL Anywhere Server - SQL Usage]
- “CREATE SEQUENCE statement” on page 655
- “DROP SEQUENCE statement” on page 771

**Standards and compatibility**
- **SQL/2008**  The ALTER SEQUENCE statement is part of optional SQL language feature T176 of the SQL/2008 standard. The CACHE clause is a vendor extension.

**Example**
The following example sets a new maximum value for a sequence named Test:

```
ALTER SEQUENCE Test
   MAXVALUE 1500;
```

---

**ALTER SERVER statement**
Modifies the attributes of a remote server.

**Syntax 1**
```
ALTER SERVER server-name
   [ CLASS server-class ]
   [ USING connection-string-info ]
   [ CAPABILITY cap-name { ON | OFF } ]
   [ CONNECTION CLOSE [ CURRENT | ALL | connection-id ] ]
```

**server-class**:
- 'ADSODBC'
- 'ASEODBC'
- 'DB2ODBC'
- 'HANAODBC'
- 'IQODBC'
- 'MIRROR'

---

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 463
connection-info-string:
  { 'data-source-name' | 'sqlanywhere-connection-string' }

Syntax 2

```
ALTER SERVER server-name
  [ CLASS 'DIRECTORY' ]
  [ USING using-string ]
  [ CAPABILITY cap-name { ON | OFF } ]
  [ CONNECTION CLOSE [ CURRENT | ALL | connection-id ] ]
```

```
using-string:
  'ROOT = path[ ;SUBDIRS = n ] [ ;READONLY = { YES | NO } ] [ ;CREATEDIRS = { YES | NO } ]
  [ ;DELIMITER = { / | \ } ]'
```

Parameters

**CLASS clause**  The CLASS clause is specified to change the server class.

**USING clause**  The USING clause is specified to change the server connection information.

The string in the USING clause can also contain local or global variable names enclosed in braces ({variable-name}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, a USING clause that contains 'DSN={@mydsn}' indicates that @mydsn is a SQL variable and that the current contents of the @mydsn variable should be substituted when a connection is made to the remote data access server.

**CAPABILITY clause**  The CAPABILITY clause turns a server capability ON or OFF. Server capabilities are stored in the ISYSCAPABILITY system table. The names of these capabilities are accessible via the SYSCAPABILITYNAME system view. The ISYSCAPABILITY system table and SYSCAPABILITYNAME system view are not populated with data until the first connection to a remote server is made. For subsequent connections, the database server's capabilities are obtained from the ISYSCAPABILITY system table.

In general, you do not need to alter a server's capabilities. It may be necessary to alter capabilities of a generic server of class ODBC.

**CONNECTION CLOSE clause (deprecated)**  This clause is deprecated. Use the DROP REMOTE CONNECTION statement.

Remarks

The ALTER SERVER statement modifies the attributes of a server. These changes do not take effect until the next connection to the remote server.
Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

Automatic commit. The CONNECTION CLOSE clause does not cause an automatic commit.

See also

- “CREATE SERVER statement” on page 657
- “DROP REMOTE CONNECTION statement” on page 768
- “DROP SERVER statement” on page 772
- “Remote data access” [SQL Anywhere Server - SQL Usage]
- “Remote servers” [SQL Anywhere Server - SQL Usage]
- “Directory access servers” [SQL Anywhere Server - SQL Usage]
- “Troubleshooting remote data access” [SQL Anywhere Server - SQL Usage]
- “SYSCAPABILITY system view” on page 1349
- “SYSCAPABILITYNAME system view” on page 1349

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example changes the server class of the Adaptive Server Enterprise server named ase_prod so its connection to SQL Anywhere is ODBC-based. Its data source name is ase_datasource.

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING 'ase_datasource';
```

The following example changes the server class of the Adaptive Server Enterprise server named ase_prod so its connection to SQL Anywhere is ODBC-based. Its data source name is obtained from the variable ase_source.

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING '{ase_source}';
CREATE VARIABLE ase_source VARCHAR(128);
SET ase_source = 'ase_datasource';
```

The following example changes a capability of a server ase_prod.

```
ALTER SERVER ase_prod
CAPABILITY 'insert select' OFF;
```

The following example alters a directory access server so that it retrieves 9 levels of subdirectories within the directory C:\Temp.

```
ALTER SERVER ase_prod
CLASS 'DIRECTORY'
USING 'ROOT=c:\temp;SUBDIRS=9';
```
ALTER SERVICE statement [HTTP web service]

Alters an existing HTTP web service.

Syntax

```
ALTER SERVICE service-name
  [ TYPE { 'RAW' | 'HTML' | 'JSON' | 'XML' } ]
  [ URL [ PATH ] { ON | OFF | ELEMENTS } ]
  [ common-attributes ]
  [ AS { statement | NULL } ]
```

```
common-attributes :
  [ AUTHORIZATION { ON | OFF } ]
  [ ENABLE | DISABLE ]
  [ METHODS 'method,...' ]
  [ SECURE { ON | OFF } ]
  [ USER { user-name | NULL } ]
```

```
method :
  DEFAULT
  POST
  GET
  HEAD
  PUT
  DELETE
  NONE
  *
```

Parameters

- **service-name**  Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

You can name your service **root**, but this name has a special function.

- **TYPE clause**  Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

  - **'RAW'**  The result set is sent to the client without any formatting. Utilization of this service requires that all content markup is explicitly provided. Complex dynamic content containing current content with markup, JavaScript and images can be generated on demand. The media type can be specified by setting the Content-Type response header using the sa_set_http_header procedure. Setting the Content-Type header to 'text/html' is good practice when generating HTML markup to ensure that all browsers display the markup as HTML and not text/plain.

  - **'HTML'**  The result set is returned as an HTML representation of a table or view.

  - **'JSON'**  The result set is returned in JavaScript Object Notation (JSON). A JSON service does not automatically process JSON input. It only presents data (in the response) in JSON format. JSON accepts POST/PUT methods where application/x-www-form-urlencoded is supported. If for a POST/PUT METHOD, Content-Type: application/json is specified, then the application may use...
http_variable('body') to retrieve the JSON (request) content. SQL Anywhere does not parse the JSON input automatically. It is up to the application to parse it. For more information about JSON, see http://www.json.org/.

- **'XML’** The result set is returned as XML. If the result set is already XML, no additional formatting is applied. Otherwise, it is automatically formatted as XML. As an alternative approach, a RAW service could return a select using the FOR XML RAW clause having set a valid Content-Type such as text/xml using sa_set_http_header procedure.

**URL clause** Determines whether URL paths are accepted and, if so, how they are processed. Specifying URL PATH has the same effect as URL.

- **OFF** Indicates that the service name in a URL request must not be followed by a path. OFF is the default setting. For example, the following form is disallowed due to the path elements /aaa/bbb/ccc.

  http://host-name/service-name/aaa/bbb/ccc

  Suppose that `CREATE SERVICE echo URL PATH OFF` was specified when creating the web service. A URL similar to `http://localhost/echo?id=1` produces the following values:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_VARIABLE('id')</td>
<td>1</td>
</tr>
<tr>
<td>HTTP_HEADER('@HttpQueryString')</td>
<td>id=1</td>
</tr>
</tbody>
</table>

- **ON** Indicates that the service name in a URL request can be followed by a path. The path value is returned by querying a dedicated HTTP variable named URL. A service can be defined to explicitly provide the URL parameter or it may be retrieved using the HTTP_VARIABLE function. For example, the following form is allowed:

  http://host-name/service-name/aaa/bbb/ccc

  Suppose that `CREATE SERVICE echo URL PATH ON` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_VARIABLE('id')</td>
<td>1</td>
</tr>
<tr>
<td>HTTP_VARIABLE('URL')</td>
<td>one/two</td>
</tr>
<tr>
<td>HTTP_HEADER('@HttpQueryString')</td>
<td>id=1</td>
</tr>
</tbody>
</table>

- **ELEMENTS** Indicates that the service name in a URL request may be followed by a path. The path is obtained in segments by specifying a single parameter keyword `URL1`, `URL2`, and so on. Each parameter may be retrieved using the HTTP_VARIABLE or NEXT_HTTP_VARIABLE functions.
These iterator functions can be used in applications where a variable number of path elements can be provided. For example, the following form is allowed:

```
http://host-name/service-name/aaa/bbb/ccc
```

Suppose that `CREATE SERVICE echo URL PATH ELEMENTS` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_VARIABLE('id')</td>
<td>1</td>
</tr>
<tr>
<td>HTTP_VARIABLE('URL1')</td>
<td>one</td>
</tr>
<tr>
<td>HTTP_VARIABLE('URL2')</td>
<td>two</td>
</tr>
<tr>
<td>HTTP_VARIABLE('URL3')</td>
<td>NULL</td>
</tr>
<tr>
<td>HTTP_HEADER('@HttpQueryString')</td>
<td>id=1</td>
</tr>
</tbody>
</table>

Up to 10 elements can be obtained. A NULL value is returned if the corresponding element is not supplied. In the above example, `HTTP_VARIABLE('URL3')` returns NULL because no corresponding element was supplied.

For more information about URLs, see “How to browse a SQL Anywhere HTTP web server” [SQL Anywhere Server - Programming] and “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming].

**AUTHORIZATION clause**  Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for all services and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) that obfuscates the user and password information using base-64 encoding. It is recommended that you use the HTTPS protocol for increased security.

**ENABLE and DISABLE clauses**  Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a 404 Not Found HTTP status.

**METHODS clause**  Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, PUT, DELETE, and NONE. An asterisk (*) may be used as a short
form to represent the POST, GET, and HEAD methods which are default request types for the RAW, HTML, and XML service types. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

<table>
<thead>
<tr>
<th>Method value</th>
<th>Applies to service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>all</td>
<td>Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.</td>
</tr>
<tr>
<td>POST</td>
<td>RAW, HTML, JSON, XML</td>
<td>Enabled by default.</td>
</tr>
<tr>
<td>GET</td>
<td>RAW, HTML, JSON, XML</td>
<td>Enabled by default.</td>
</tr>
<tr>
<td>HEAD</td>
<td>RAW, HTML, JSON, XML</td>
<td>Enabled by default.</td>
</tr>
<tr>
<td>PUT</td>
<td>RAW, HTML, JSON, XML</td>
<td>Not enabled by default.</td>
</tr>
<tr>
<td>DELETE</td>
<td>RAW, HTML, JSON, XML</td>
<td>Not enabled by default.</td>
</tr>
<tr>
<td>NONE</td>
<td>all</td>
<td>Use NONE to disable access to a service.</td>
</tr>
<tr>
<td>*</td>
<td>RAW, HTML, JSON, XML</td>
<td>Same as specifying 'POST,GET,HEAD'.</td>
</tr>
</tbody>
</table>

For example, you can use either of the following clauses to specify that a service supports all HTTP method types:

```
    METHODS '*,PUT,DELETE'
    METHODS 'POST,GET,HEAD,PUT,DELETE'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
    METHODS 'DEFAULT'
```

**SECURE clause**  
Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

**USER clause**  
Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service
requiring authorization results in a 401 Authorization Required HTTP response status. Based on this response, the web browser prompts for a user ID and password.

Caution
It is strongly recommended that you specify a USER clause when authorization is enabled (default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

statement  Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

An HTTP request to a non-DISH service with no statement specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use statement to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following statement demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

    CALL AuthenticateUser ( :user_name, :user_password );

For more information about passing host variables to a function or procedure, see “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming].

Remarks
The ALTER SERVICE statement modifies the attributes of a web service.

Privileges
You must be the owner of the service, or have the MANAGE ANY WEB SERVICE system privilege.

Side effects
None.
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example demonstrates how to disable an existing web service using the ALTER SERVICE statement:

```sql
CREATE SERVICE WebServiceTable
  TYPE 'RAW'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT *
    FROM SYS.SYSTAB;

ALTER SERVICE WebServiceTable DISABLE;
```

**ALTER SERVICE statement [SOAP web service]**

Alters an existing HTTP over SOAP or DISH service.

**Syntax 1 - SOAP over HTTP services**

```sql
ALTER SERVICE service-name
  [ TYPE 'SOAP' ]
  [ DATATYPE { ON | OFF | IN | OUT } ]
  [ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL } ]
  [ common-attributes ]
  [ AS statement ]

common-attributes :
  [ AUTHORIZATION { ON | OFF } ]
  [ ENABLE | DISABLE ]
  [ METHODS 'method,...' ]
```
Syntax 2 - DISH services

ALTER SERVICE service-name
  TYPE 'DISH'
  [ GROUP { group-name | NULL } ]
  [ FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL } ]
  [ common-attributes ]

Parameters

The descriptions of the ALTER SERVICE clauses are identical to those of the CREATE SERVICE statement.

Parameters

service-name  Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

Unlike other services, you cannot use a slash (/) anywhere in a DISH service name.

You can name your service root, but this name has a special function.

TYPE clause  Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

- SOAP'  The result set is returned as an XML payload known as a SOAP envelope. The format of the data may be further refined using by the FORMAT clause. A request to a SOAP service must be a valid SOAP request, not just a general HTTP request. For more information about the SOAP standards, see http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.
A DISH service (Determine SOAP Handler) is a SOAP endpoint that references any SOAP service within its GROUP context. It also exposes the interfaces to its SOAP services by generating a WSDL (Web Services Description Language) for consumption by SOAP client toolkits.

GROUP clause  A DISH service without a GROUP clause exposes all SOAP services defined within the database. By convention, the SOAP service name can be composed of a GROUP and a NAME element. The name is delimited from the group by the last slash character. For example, a SOAP service name defined as 'aaa/bbb/ccc' is 'ccc', and the group is 'aaa/bbb'. Delimiting a DISH service using this convention is invalid. Instead, a GROUP clause is applied to specify the group of SOAP services for which it is to be the SOAP endpoint.

Note
Slashes are converted to underscores within the WSDL to produce valid XML. Use caution when using a DISH service that does not specify a GROUP clause such that it exposes all SOAP services that may contain slashes. Use caution when using groups with SOAP service names that contain underscores to avoid ambiguity.

DATATYPE clause  Applies to SOAP services only. When DATATYPE OFF is specified, SOAP input parameters and response data are defined as XMLSchema string types. In most cases, true data types are preferred because it negates the need for the SOAP client to cast the data prior to computation. Parameter data types are exposed in the schema section of the WSDL generated by the DISH service. Output data types are represented as XML schema type attributes for each column of data.

The following values are permitted for the DATATYPE clause:

- **ON** Generates data typing of input parameters and result set responses.
- **OFF** All input parameters and response data are typed as XMLSchema string (default).
- **IN** Generates true data types for input parameters only. Response data types are XMLSchema string.
- **OUT** Generates true data types for responses only. Input parameters are typed as XMLSchema string.

For more information about SOAP services, see “Tutorial: Using SQL Anywhere to access a SOAP/DISH service” [SQL Anywhere Server - Programming].

For more information about mapping XMLSchema types to SQL data types, see “SOAP data types” [SQL Anywhere Server - Programming].

FORMAT clause  This clause specifies the output format when sending responses to SOAP client applications.

The SOAP service format is dictated by the associated DISH service format specification when it is not specified by the SOAP service. The default format is DNET.

SOAP requests should be directed to the DISH service (the SQL Anywhere SOAP endpoint) to leverage common formatting rules for a group of SOAP services (SOAP operations). A SOAP service FORMAT specification overrides that of a DISH service. The format specification of the DISH service is used when
a SOAP service does not define a FORMAT clause. If no FORMAT is provided by either service then the default is 'DNET'.

The following formats are supported:

- **'DNET'**  The output is in a System.Data.DataSet compatible format for consumption by .NET client applications. (default)

- **'CONCRETE'**  This output format is used to support client SOAP toolkits that are capable of generating interfaces representing arrays of row and column objects but are not able to consume the DNET format. Java and .NET clients can easily consume this output format.

  The specific format is exposed within the WSDL as an explicit dataset object or a SimpleDataset. Both dataset representations describe a data structure representing an array of rows with each row containing an array of columns. An explicit dataset object has the advantage of representing the actual shape of the result set by providing parameter names and datatypes for each column in the row. In contrast, the SimpleDataset exposes rows containing an unbounded number of columns of any type.

  FORMAT 'CONCRETE' EXPLICIT ON requires that the Service statement calls a stored procedure which defines a RESULT clause. Having met this condition, the SOAP service will expose an explicit dataset whose name begins with the service name appended with Dataset.

  If the condition is not met, a SimpleDataset is used.

- **'XML'**  The output is generated in an XMLSchema string format. The response is an XML document that requires further processing by the SOAP client to extract column data. This format is suitable for SOAP clients that cannot generate intermediate interface objects that represent arrays of rows and columns.

- **NULL**  A NULL type causes the SOAP or DISH service to use the default behavior. The format type of an existing service is overwritten when using the NULL type in an ALTER SERVICE statement.

**AUTHORIZATION clause**  Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for SOAP services, and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

  When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) which obfuscates the user and password information using base-64 encoding. It is recommended that you use the HTTPS protocol for increased security.

**ENABLE and DISABLE clauses**  Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a 404 Not Found HTTP status.
METHODS clause  Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods. The default method types for SOAP services are POST and HEAD. The default method types for DISH services are GET, POST, and HEAD. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

<table>
<thead>
<tr>
<th>Method value</th>
<th>Applies to service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>both</td>
<td>Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.</td>
</tr>
<tr>
<td>POST</td>
<td>both</td>
<td>Enabled by default for SOAP.</td>
</tr>
<tr>
<td>GET</td>
<td>DISH only</td>
<td>Enabled by default for DISH.</td>
</tr>
<tr>
<td>HEAD</td>
<td>both</td>
<td>Enabled by default for SOAP and DISH.</td>
</tr>
<tr>
<td>NONE</td>
<td>both</td>
<td>Use NONE to disable access to a service. When applied to a SOAP service, the service cannot be directly accessed by a SOAP request. This enforce exclusive access to a SOAP operation through a DISH service SOAP endpoint. It is recommended that you specify METHOD NONE for each SOAP service.</td>
</tr>
<tr>
<td>*</td>
<td>DISH only</td>
<td>Same as specifying 'POST,GET,HEAD'.</td>
</tr>
</tbody>
</table>

For example, you can use the following clause to specify that a service supports all SOAP over HTTP method types:

```
METHODS 'POST,HEAD'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

SECURE clause  Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

USER clause  Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service
requiring authorization results in a 401 Authorization Required HTTP response status. Based on this response, the web browser prompts for a user ID and password.

**Caution**

It is strongly recommended that you specify a USER clause when authorization is enabled (the default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

**statement** Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

A DISH service is the only service that must either define a null statement, or not define a statement. A SOAP service must define a statement. Any other SERVICE can have a NULL statement, but only if it is configured with AUTHORIZATION ON.

An HTTP request to a non-DISH service with no statement specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use statement to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following statement demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

For more information about passing host variables to a function or procedure, see “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming].

**Remarks**

The ALTER SERVICE statement modifies the attributes of a web service.

**Privileges**

You must be the owner of the service, or have the MANAGE ANY WEB SERVICE system privilege.

**Side effects**

None.
See also

- “CREATE SERVICE statement [SOAP web service]” on page 668
- “DROP SERVICE statement” on page 772
- “Tutorial: Using SQL Anywhere to access a SOAP/DISH service” [SQL Anywhere Server - Programming]
- “SYSWEBSERVICE system view” on page 1418
- “-xs database server option” [SQL Anywhere Server - Database Administration]
- “SQL Anywhere as an HTTP web server” [SQL Anywhere Server - Programming]
- “How to create and customize a root web service” [SQL Anywhere Server - Programming]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example demonstrates how to disable an existing web service using the ALTER SERVICE statement:

```
CREATE SERVICE WebServiceTable
  TYPE 'SOAP'
  AUTHORIZATION OFF
  USER DBA
  AS SELECT *
       FROM SYS.SYSTAB;

ALTER SERVICE WebServiceTable DISABLE;
```

**ALTER SPATIAL REFERENCE SYSTEM statement**

Changes the settings of an existing spatial reference system. See the Remarks section for considerations before altering a spatial reference system.

**Syntax**

```
ALTER SPATIAL REFERENCE SYSTEM
srs-name
[ srs-attribute [ srs-attribute ... ] ]

srs-name : string

srs-attribute :
  SRID srs-id
  | DEFINITION { definition-string | NULL }
  | ORGANIZATION { organization-name IDENTIFIED BY organization-srs-id | NULL }
  | TRANSFORM DEFINITION { transform-definition-string | NULL }
  | LINEAR UNIT OF MEASURE linear-unit-name
  | ANGULAR UNIT OF MEASURE { angular-unit-name | NULL }
  | TYPE { ROUND EARTH | PLANAR }
  | COORDINATE coordinate-name { UNBOUNDED | BETWEEN low-number AND high-number }
  | ELLIPSOID SEMI MAJOR AXIS semi-major-axis-length { SEMI MINOR AXIS semi-minor-axis-length | INVERSE FLATTENING inverse-flattening-ratio }
  | SNAP TO GRID { grid-size | DEFAULT }
  | TOLERANCE { tolerance-distance | DEFAULT }
```
POLYGON FORMAT  polygon-format
STORAGE FORMAT  storage-format

srs-id : integer

semi-major-axis-length : number

semi-minor-axis-length : number

inverse-flattening-ratio : number

grid-size : DOUBLE : usually between 0 and 1

tolerance-distance : number

axis-order : { 'x/y/z/m' | 'long/lat/z/m' | 'lat/long/z/m' }

polygon-format : { 'CounterClockWise' | 'Clockwise' | 'EvenOdd' }

storage-format : { 'Internal' | 'Original' | 'Mixed' }

Parameters

Complete definitions for each of the clauses is provided in the CREATE SPATIAL REFERENCE SYSTEM statement.

IDENTIFIED BY clause     Use this clause to change the SRID number for the spatial reference system.

DEFINITION clause     Use this clause to set, or override, default coordinate system settings.

ORGANIZATION clause     Use this clause to specify information about the organization that created the spatial reference system that the spatial reference system is based on.

TRANSFORM DEFINITION clause     Use this clause to specify a description of the transform to use for the spatial reference system. Currently, only the PROJ.4 transform is supported.

The transform definition is used by the ST_Transform method when transforming data between spatial reference systems. Some transforms may still be possible even if there is no transform-definition-string defined.

COORDINATE clause     Use this clause to specify the bounds on the spatial reference system's dimensions. coordinate-name is the name of the coordinate system used by the spatial reference system. For non-geographic types coordinate-name can be x, y, or m. For geographic types, coordinate-name can be LATITUDE, LONGITUDE, z, or m.

LINEAR UNIT OF MEASURE clause     Use this clause to specify the linear unit of measure for the spatial reference system. The value you specify must match a linear unit of measure defined in the ST_UNITS_OF_MEASURE system view.

ANGULAR UNIT OF MEASURE clause     Use this clause to specify the angular unit of measure for the spatial reference system. The value you specify must match an angular unit of measure defined in the ST_UNITS_OF_MEASURE system table.
**TYPE clause** Use the TYPE clause to control how the spatial reference system interprets lines between points. For geographic spatial reference systems, the TYPE clause can specify either ROUND EARTH (the default) or PLANAR. For non-geographic spatial reference systems, the type must be PLANAR.

**ELLIPSOID clause** Use the ellipsoid clause to specify the values to use for representing the Earth as an ellipsoid for spatial reference systems of type ROUND EARTH. If the DEFINITION clause is present, it can specify ellipsoid definition. If the ELLIPSOID clause is specified, it overrides this default ellipsoid.

**SNAP TO GRID clause** For flat-Earth (planar) spatial reference systems, use the SNAP TO GRID clause to define the size of the grid SQL Anywhere uses when performing calculations. Specify SNAP TO GRID DEFAULT to set the grid size to the default that the database server would use.

For round-Earth spatial reference systems, SNAP TO GRID must be set to 0.

**TOLERANCE clause** For flat-Earth (planar) spatial reference systems, use the TOLERANCE clause to specify the precision to use when comparing points.

For round-Earth spatial reference systems, TOLERANCE must be set to 0.

**POLYGON FORMAT clause** Use the POLYGON FORMAT clause to change the polygon interpretation. The following values are supported:

- 'CounterClockwise'
- 'Clockwise'
- 'EvenOdd'

The default polygon format is 'EvenOdd'.

**STORAGE FORMAT clause** Use the STORAGE FORMAT clause to control what is stored when spatial data is loaded into the database. Possible values are:

- 'Internal' SQL Anywhere stores only the normalized representation. Specify this when the original input characteristics do not need to be reproduced. This is the default for planar spatial reference systems (TYPE PLANAR).

  **Note** If you are using MobiLink to synchronize your spatial data, you should specify **Mixed** instead. MobiLink tests for equality during synchronization, which requires the data in its original format.

- 'Original' SQL Anywhere stores only the original representation. The original input characteristics can be reproduced, but all operations on the stored values must repeat normalization steps, possibly slowing down operations on the data.

- 'Mixed' SQL Anywhere stores the internal version and, if it is different from the original version, SQL Anywhere stores the original version as well. By storing both versions, the original representation characteristics can be reproduced and operations on stored values do not need to repeat normalization steps. However, storage requirements may increase significantly because potentially two representations are being stored for each geometry.

  Mixed is the default format for round-Earth spatial reference systems (TYPE ROUND EARTH).
Remarks
You cannot alter a spatial reference system if there is existing data that references it. For example, if you have a column declared as ST_Point(SRID=8743), you cannot alter the spatial reference system with SRID 8743. This is because many spatial reference system attributes, such as storage format, impact the storage format of the data. If you have data that references the SRID, create a new spatial reference system and transform the data to the new SRID.

Privileges
You must be the owner of the spatial reference system, or have one of the following privileges:

- ALTER privilege on the spatial reference system
- MANAGE ANY SPATIAL OBJECT system privilege
- ALTER ANY OBJECT system privilege

Side effects
None

See also
- “CREATE SPATIAL REFERENCE SYSTEM statement” on page 673
- “DROP SPATIAL REFERENCE SYSTEM statement” on page 773
- “Spatial data” [SQL Anywhere Server - Spatial Data Support]
- “ST_Transform method for type ST_Geometry” [SQL Anywhere Server - Spatial Data Support]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example changes the polygon format of a fictitious spatial reference system named mySpatialRef to EvenOdd.

```
ALTER SPATIAL REFERENCE SYSTEM mySpatialRef
POLYGON FORMAT 'EvenOdd';
```

ALTER STATISTICS statement
Controls whether statistics are automatically updated on a column, or columns, in a table.

Syntax
```
ALTER STATISTICS
  [ ON ]
  [ ( column1 [, column2 ... ] ) ]
AUTO UPDATE { ENABLE | DISABLE }
```

Parameters
- ON The word ON is optional. Including it has no impact on the execution of the statement.
- AUTO UPDATE clause Specify whether to enable or disable automatic updating of statistics for the column(s).
Remarks
During normal execution of queries, DML statements, and LOAD TABLE statements, the database server automatically maintains column statistics for use by the optimizer. The benefit of maintaining statistics for some columns may not outweigh the overhead necessary to generate them. For example, if a column is not queried often, or if it is subject to periodic mass changes that are eventually rolled back, there is little value in continually updating its statistics. Use the ALTER STATISTICS statement to suppress the automatic updating of statistics for these types of columns.

When automatic updating is disabled, you can still update the statistics for the column using the CREATE STATISTICS and DROP STATISTICS statements. However, you should only update them if it has been determined that it would have a positive impact on performance. Normally, column statistics should not be disabled.

Privileges
You must be the table owner, or have one of the following privileges:

● MANAGE ANY STATISTICS system privilege
● ALTER ANY OBJECT system privilege

Side effects
If automatic updating has been disabled, the statistics may become out of date. Re-enabling does not immediately bring them up to date. Execute the CREATE STATISTICS statement to recreate them, if necessary.

See also
● “CREATE STATISTICS statement” on page 682
● “DROP STATISTICS statement” on page 775

Standards and compatibility
● SQL/2008 Vendor extension.

Example
The following example disables the automatic updating of statistics on the Street column in the Customers table:

ALTER STATISTICS GROUPO.Customers ( Street ) AUTO UPDATE DISABLE;

ALTER SYNCHRONIZATION PROFILE statement
[MobiLink]
Changes a SQL Anywhere synchronization profile. Synchronization profiles are named collections of synchronization options that can be used to control synchronization.

Syntax

ALTER SYNCHRONIZATION PROFILE name
MERGE string
Parameters

**name**  The name of the synchronization profile to alter.

**MERGE clause**  Use this clause to change existing, or add new, options to a synchronization profile.

**string**  A string of one or more synchronization option value pairs, separated by semicolons. For example, 'option1=value1;option2=value2'.

Remarks

Synchronization profiles define how a SQL Anywhere database synchronizes with the MobiLink server.

When MERGE is used in the ALTER SYNCHRONIZATION PROFILE statement, options specified in the string are added to those already in the synchronization profile. If an option in the string already exists in profile, then the value from the string replaces the value already stored in the profile.

For example, executing the following statements leaves the profile `myProfile` with the value `subscription=s2;verbosity=high;uploadonly=on`.

```
CREATE SYNCHRONIZATION PROFILE myProfile 'subscription=p1;verbosity=high';
ALTER SYNCHRONIZATION PROFILE myProfile MERGE 'subscription=p2;uploadonly=on';
```

When setting extended options, use the following syntax:

```
ALTER SYNCHRONIZATION PROFILE myprofile MERGE 's=mysub;e={ctp=tcpip;adr=''host=localhost;port=2439''}';
```

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

See also

-  “CREATE SYNCHRONIZATION PROFILE statement [MobiLink]” on page 685
-  “DROP SYNCHRONIZATION PROFILE statement [MobiLink]” on page 778

Standards and compatibility

-  SQL/2008  Vendor extension.

**ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]**

Alters the properties of a synchronization subscription in a SQL Anywhere remote database.

Syntax

```
ALTER SYNCHRONIZATION SUBSCRIPTION
{ subscription-name | TO publication-name [ FOR ml-username, ... ] } { alter-clause ... }
```
alter-clause:
RENAME new-subscription-name
| TYPE network-protocol
| ADDRESS protocol-options
| ADD OPTION option=VALUE, ...
| ALTER OPTION option=VALUE, ...
| DELETE { ALL OPTION | OPTION option, ... }
| SET SCRIPT VERSION=script-version

subscription-name: identifier
publication-name: identifier
ml-username: identifier
new-subscription-name: identifier

network-protocol: http | https | tls | tcpip | NULL
protocol-options: string | NULL
value: string | integer
option: identifier

script-version: string | NULL

Parameters

TO clause  This clause specifies the name of a publication.

If the TO clause is used without a FOR clause, you cannot use the RENAME or SET SCRIPT VERSION clauses.

FOR clause  This clause specifies one or more MobiLink user names.

Omit the FOR clause to set the protocol type, protocol options, and extended options for a publication.

If the TO clause is used without a FOR clause, you cannot use the RENAME or SET SCRIPT VERSION clauses.

RENAME clause  This clause specifies a new name for the subscription.

If the TO clause is used without a FOR clause, you cannot use the RENAME clause.

TYPE clause  This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause  This clause specifies network protocol options, including the location of the MobiLink server.

ADD OPTION, ALTER OPTION, DELETE OPTION, and DELETE ALL OPTION clauses  These clauses allow you to add, alter, delete, or delete all extended options. You can specify only one option in each clause. No option is specified for Delete All.
The values for each option cannot contain the characters " = " or ", " or "; ".

**SET SCRIPT VERSION clause**  This clause specifies the script version to use during synchronization. You can alter the script version without making a schema change.

If the TO clause is used without a FOR clause, you cannot use the SET SCRIPT VERSION clause.

* Cloud note: You can specify NULL for the script-version value for cloud tenant databases.

**Remarks**

The network-protocol, protocol-options, and options can be set in several places.

This statement causes options and other information to be stored in the SQL Anywhere ISYSSYNC system table. Depending on the privileges a user has, they may be able to view the information, which could include passwords and encryption certificates. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

**Privileges**

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

**Side effects**

Automatic commit.

**See also**

- “CREATE PUBLICATION statement [MobiLink] [SQL Remote]” on page 647
- “DROP PUBLICATION statement [MobiLink] [SQL Remote]” on page 766
- “CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” on page 686
- SQL Anywhere MobiLink clients: “Synchronization subscription creation” [MobiLink - Client Administration]
- UltraLite MobiLink clients: “UltraLite client synchronization design” [UltraLite - Database Management and Reference]
- “SYSSYNC system view” on page 1396
- “MobiLink client network protocol options” [MobiLink - Client Administration]
- “MobiLink SQL Anywhere client extended options” [MobiLink - Client Administration]
- “CommunicationType (ctp) extended option” [MobiLink - Client Administration]
- “dbmlsync syntax” [MobiLink - Client Administration]
- “Priority order” [MobiLink - Client Administration]
- “Script versions” [MobiLink - Server Administration]

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following example changes the address of the MobiLink server for the sales subscription:

```
ALTER SYNCHRONIZATION SUBSCRIPTION sales
  TYPE TCPIP
  ADDRESS 'host=10.11.12.132;port=2439';
```
ALTER SYNCHRONIZATION USER statement [MobiLink]

Alters the properties of a MobiLink user in a SQL Anywhere remote database.

Syntax

```
ALTER SYNCHRONIZATION USER ml-username
  [ TYPE network-protocol ]
  [ ADDRESS protocol-options ]
  [ ADD OPTION option=value, ... ]
  [ ALTER OPTION option=value, ... ]
  [ DELETE { ALL OPTION | OPTION option } ]
```

- `ml-username`: identifier
- `network-protocol`: `http` | `https` | `tls` | `tcpip` | `NULL`
- `protocol-options`: `string` | `NULL`
- `value`: `string` | `integer`

Parameters

- **TYPE clause**  This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

- **ADDRESS clause**  This clause specifies `protocol-options` in the form `keyword=value`, separated by semicolons. Which settings you supply depends on the communication protocol you are using (TCP/IP, TLS, HTTP, or HTTPS).

- **ADD OPTION, ALTER OPTION, DELETE OPTION, and DELETE ALL OPTION clauses**  These clauses allow you to add, modify, delete, or delete all extended options. You may specify only one option in each clause. No option is specified for Delete All.

Remarks

The `network-protocol, protocol-options, and options` can be set in several places.

This statement causes options and other information to be stored in the SQL Anywhere ISYSSYNC system table. Depending on the privileges a user has, passwords and encryption certificates can be visible. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

Requires exclusive access to all tables referred to in the publication.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.
ALTER TABLE statement

Modifies a table definition or disables dependent views.

Syntax 1 - Altering an existing table

```
ALTER TABLE [owner.]table-name { alter-clause, ... }
```

```
alter-clause :
  ADD create-clause
  | ALTER column-name column-alteration
  | ALTER [ CONSTRAINT constraint-name ] CHECK ( condition )
  | DROP drop-object

  RENAME rename-object
  | table-alteration

create-clause :
  column-name [ AS ] column-data-type [ new-column-attribute ... ]
  | table-constraint
  | PCTFREE integer

column-alteration :
  { column-data-type | alterable-column-attribute } [ alterable-column-attribute ... ]
  | SET COMPUTE ( compute-expression )
  | ADD [ constraint-name ] CHECK ( condition )
  | DROP { DEFAULT | COMPUTE | CHECK | CONSTRAINT constraint-name }

drop-object :
  column-name
  | CHECK
  | CONSTRAINT constraint-name
  | UNIQUE [ CLUSTERED ] ( index-columns-list )
  | FOREIGN KEY fkey-name
  | PRIMARY KEY

rename-object :
  new-table-name
```
 COLUMN-NAME TO new-column-name
 CONSTRAINT constraint-name TO new-constraint-name

 table-alteration:
 PCTFREE DEFAULT
 [ NO | NOT ] ENCRYPTED

 new-column-attribute:
 [ NOT ] NULL
 DEFAULT default-value
 COMPRESSED
 INLINE { inline-length | USE DEFAULT }
 PREFIX { prefix-length | USE DEFAULT }
 [ NO ] INDEX
 IDENTITY
 COMPUTE ( expression )
 column-constraint

 table-constraint:
 [ CONSTRAINT constraint-name ] {
  CHECK ( condition )
  | UNIQUE [ CLUSTERED | NONCLUSTERED ] ( column-name [ ASC | DESC ], ... )
  | PRIMARY KEY [ CLUSTERED | NONCLUSTERED ] ( column-name [ ASC | DESC ], ... )
  | foreign-key
 }

 column-constraint:
 [ CONSTRAINT constraint-name ] {
  CHECK ( condition )
  | UNIQUE [ CLUSTERED | NONCLUSTERED ] [ ASC | DESC ]
  | PRIMARY KEY [ CLUSTERED | NONCLUSTERED ] [ ASC | DESC ]
  | REFERENCES table-name [ ( column-name ) ]
  | [ MATCH [ UNIQUE ] { SIMPLE | FULL } ]
  | [ actions ] CLUSTERED | NONCLUSTERED
  | NOT NULL
 }

 alterable-column-attribute:
 [ NOT ] NULL
 DEFAULT default-value
 [ CONSTRAINT constraint-name ] CHECK { NULL | ( condition ) }
 [ NOT | NOT ] COMPRESSED
 INLINE { inline-length | USE DEFAULT }
 PREFIX { prefix-length | USE DEFAULT }
 [ NO ] INDEX

 default-value:
 special-value
 string
 global variable
 [ - ] number
 ( constant-expression )
 ( sequence-expression )
 built-in-function( constant-expression )
 AUTOINCREMENT
 GLOBAL AUTOINCREMENT [ ( partition-size ) ]
special-value :
  CURRENT DATABASE
  CURRENT DATE
  CURRENT TIME
  [ CURRENT ] TIMESTAMP
  CURRENT PUBLISHER
  CURRENT REMOTE USER
  [ CURRENT ] USER
  [ CURRENT ] UTC TIMESTAMP
  LAST USER
  NULL

foreign-key :
  [ NOT NULL ] FOREIGN KEY [ role-name ]
  [ ( column-name [ ASC | DESC ], ... )
  REFERENCES table-name
  [ ( pkey-column-list ) ]
  [ MATCH [ UNIQUE ] { SIMPLE | FULL } ]
  [ actions ] [ CHECK ON COMMIT ] [ CLUSTERED ]
  [ FOR OLAP WORKLOAD ]

actions :
  [ ON UPDATE action ] [ ON DELETE action ]

action :
  CASCADE | SET NULL | SET DEFAULT | RESTRICT

Syntax 2 - Disabling view dependencies
  ALTER TABLE [owner.]table-name {
    DISABLE VIEW DEPENDENCIES
  }

Syntax 3 - Altering a table owner
  ALTER TABLE [owner.]table-name ALTER OWNER TO owner
  [ { PRESERVE | DROP } PRIVILEGES ]
  [ { PRESERVE | DROP } FOREIGN KEYS ]

Parameters

Adding clauses  The following section explains the clauses used for adding columns or table constraints to a table:

ADD column-name [ AS ] column-data-type [ new-column-attribute ... ] clause  Use this clause to add a new column to the table, specifying the data type and attributes for the column.

NULL and NOT NULL clauses  Use this clause to specify whether to allow NULLs in the column. By default, new columns allow NULL values. BIT type columns automatically have the NOT NULL constraint applied when they are created, but you can declare a BIT type column to be nullable.

DEFAULT clause  If a DEFAULT value is specified, it is used as the value for the column in any INSERT statement that does not specify a value for the column. If no DEFAULT value is specified, it is equivalent to DEFAULT NULL.

Following is a list of possible values for DEFAULT:
You use one of several specials values in the DEFAULT clause.

**[ CURRENT ] TIMESTAMP**  Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT TIMESTAMP, a default value is provided for inserts, and the value is updated with the current date and time whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT TIMESTAMP instead of DEFAULT TIMESTAMP.

Columns declared with DEFAULT TIMESTAMP contain unique values, so that applications can detect near-simultaneous updates to the same row. If the current TIMESTAMP value is the same as the last value, it is incremented by the value of the default_timestamp_increment option.

You can automatically truncate TIMESTAMP values in SQL Anywhere based on the default_timestamp_increment option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

The global variable @@dbts returns a TIMESTAMP value representing the last value generated for a column using DEFAULT TIMESTAMP.

**[ CURRENT ] UTC TIMESTAMP**  Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT UTC TIMESTAMP, a default value is provided for inserts, and the value is updated with the current Coordinated Universal Time (UTC) whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT UTC TIMESTAMP instead of DEFAULT UTC TIMESTAMP.

The behavior of this default is the same as TIMESTAMP and CURRENT TIMESTAMP except that the date and time of day is in Coordinated Universal Time (UTC).

**string**  See “Strings” on page 6.

**global-variable**  See “Global variables” on page 82.

**constant-expression**  Constant expressions that do not reference database objects are allowed in a DEFAULT clause, so functions such as GETDATE or DATEADD can be used. If the expression is not a function or simple value, it must be enclosed in parentheses.

**sequence-expression**  You can set DEFAULT to the current value or next value from a sequence in the database.

**AUTOINCREMENT**  When using AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type.

On inserts into the table, if a value is not specified for the AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column that is larger than the current maximum value for the column, that value is inserted and then used as a starting point for subsequent inserts.
Deleting rows does not decrement the AUTOINCREMENT counter. Gaps created by deleting rows can only be filled by explicit assignment when using an insert. After an explicit insert of a column value less than the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the max_identity column in the SYSTABCOL system view. When the next value to be generated exceeds the maximum value that can be stored in the column to which the AUTOINCREMENT is assigned, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated.

The next value to use for a column can be reset using the sa_reset_identity procedure.

**GLOBAL AUTOINCREMENT**  This default is intended for use when multiple databases are used in a MobiLink synchronization environment or SQL Remote replication.

This option is similar to AUTOINCREMENT, except that the domain is partitioned. Each partition contains the same number of values. You assign each copy of the database a unique global database identification number. SQL Anywhere supplies default values in a database only from the partition uniquely identified by that database's number.

The partition size can be specified in parentheses immediately following the AUTOINCREMENT keyword. The partition size can be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, be exhausted.

If the column is of type BIGINT or UNSIGNED BIGINT, the default partition size is $2^{32} = 4294967296$; for columns of all other types, the default partition size is $2^{16} = 65536$. Since these defaults may be inappropriate, especially if your column is not of type INT or BIGINT, it is best to specify the partition size explicitly.

When using this default, the value of the public option global_database_id in each database must be set to a unique, non-negative integer. This value uniquely identifies the database and indicates from which partition default values are to be assigned. The range of allowed values is $np + 1$ to $p(n + 1)$, where $n$ is the value of the public option global_database_id and $p$ is the partition size. For example, if you define the partition size to be 1000 and set global_database_id to 3, then the range is from 3001 to 4000.

If the previous value is less than $p(n + 1)$, the next default value is one greater than the previous largest value in the column. If the column contains no values, the first default value is $np + 1$. Default column values are not affected by values in the column outside the current partition; that is, by numbers less than $np + 1$ or greater than $p(n + 1)$. Such values may be present if they have been replicated from another database via MobiLink or SQL Remote.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

GLOBAL AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the max_identity column in the SYSTABCOL system view. When the supply of values
within the partition has been exhausted, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated. In this case, a new value of global_database_id should be assigned to the database to allow default values to be chosen from another partition. To detect that the supply of unused values is low and handle this condition, create an event of type GlobalAutoincrement.

Because the public option global_database_id cannot be set to a negative value, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

If the public option global_database_id is set to the default value of 2147483647, a NULL value is inserted into the column. If NULL values are not permitted, attempting to insert the row causes an error.

The next value to use for a column can be reset using the sa_reset_identity procedure.

- **LAST USER**  
  LAST USER is the user ID of the user who last modified the row.

  LAST USER can be used as a default value in columns with character data types.

  On INSERT, this default has the same effect as CURRENT USER.

  On UPDATE, if a column with a default of LAST USER is not explicitly modified, it is changed to the name of the current user.

  When used with DEFAULT TIMESTAMP or DEFAULT UTC TIMESTAMP, a default of LAST USER can be used to record (in separate columns) both the user and the date and time a row was last changed.

**column-constraint clause**  
Use this clause to add a constraint to the column.

**Note**  
With the exception of CHECK constraints, when a new constraint is added, the database server validates existing values to confirm that they satisfy the constraint. CHECK constraints are enforced only for operations that occur after the table alteration is complete.

Possible column constraints include:

- **CHECK clause**  
  This constraint allows arbitrary conditions to be verified. For example, a CHECK constraint could be used to ensure that a column called Sex only contains the values M or F.

  If you need to create a CHECK constraint that involves a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

- **UNIQUE clause**  
  Use this subclause to specify that values in the column must be unique, and whether to create a clustered or nonclustered index.

- **PRIMARY KEY clause**  
  Use this subclause to make the column a primary key, and specify whether to use a clustered index.
• **REFERENCES clause**  Use this subclause to add or alter a reference to another table, to specify how matches are handled, and to specify whether to use a clustered index.

• **MATCH clause**  Use this subclause to control what is considered a match when using a multi-column foreign key. It also allows you to specify uniqueness for the key, thereby eliminating the need to declare uniqueness separately.

• **NULL and NOT NULL clauses**  Use this clause to specify whether to allow NULL values in the column. By default, NULLs are allowed.

**COMPRESSED clause**  Use this clause to compress the column.

**INLINE and PREFIX clauses**  The INLINE clause specifies the maximum BLOB size, in bytes, to store within the row. BLOBs smaller than or equal to the value specified by the INLINE clause are stored within the row. BLOBs that exceed the value specified by the INLINE clause are stored outside the row in table extension pages. Also, a copy of some bytes from the beginning of the BLOB may be kept in the row when a BLOB is larger than the INLINE value. Use the PREFIX clause to specify how many bytes are kept in the row. The PREFIX clause can improve the performance of requests that need the prefix bytes of a BLOB to determine if a row is accepted or rejected.

The prefix data for a compressed column is stored uncompressed, so if all the data required to satisfy a request is stored in the prefix, no decompression is necessary.

If neither INLINE nor PREFIX is specified, or if USE DEFAULT is specified, default values are applied as follows:

• For character data type columns, such as CHAR, NCHAR, and LONG VARCHAR, the default value of INLINE is 256, and the default value of PREFIX is 8.

• For binary data type columns, such as BINARY, LONG BINARY, VARBINARY, BIT, VARBIT, LONG VARBIT, BIT VARYING, and UUID, the default value of INLINE is 256, and the default value of PREFIX is 0.

**Note**
It is strongly recommended that you use the default values unless there are specific circumstances that require a different setting. The default values have been chosen to balance performance and disk space requirements. For example, if you set INLINE to a large value, and all the BLOBs are stored inline, row processing performance may degrade. If you set PREFIX too high, you increase the amount of disk space required to store BLOBs since the prefix data is a duplicate of a portion of the BLOB.

If only one of the values is specified, the other value is automatically set to the largest amount that does not conflict with the specified value. Neither the INLINE nor PREFIX value can exceed the database page size. Also, there is a small amount of overhead reserved in a table page that cannot be used to store row data. Therefore, specifying an INLINE value approximate to the database page size can result in a slightly smaller number of bytes being stored inline.

**INDEX and NO INDEX clauses**  When storing BLOBs (character or binary types only), specify INDEX to create BLOB indexes on inserted values that exceed the internal BLOB size threshold (approximately eight database pages). This is the default behavior.
BLOB indexes can improve performance when random access searches within the BLOBs are required. However, for some types of BLOB values, such as images and multimedia files that will never require random-access, performance can improve if BLOB indexing is turned off. To turn off BLOB indexing for a column, specify NO INDEX.

Note
A BLOB index is not the same as a table index. A table index is created to index values in one or more columns.

IDENTITY clause  IDENTITY is a Transact-SQL-compatible alternative to using DEFAULT AUTOINCREMENT. In SQL Anywhere, a column defined with IDENTITY is implemented as DEFAULT AUTOINCREMENT.

COMPUTE clause  When a column is created using a COMPUTE clause, its value in any row is the value of the supplied expression. Columns created with this constraint are read-only columns for applications: the value is changed by the database server whenever the row is modified. The COMPUTE expression should not return a non-deterministic value. For example, it should not include a special value such as CURRENT_TIMESTAMP, or a non-deterministic function. If a COMPUTE expression returns a non-deterministic value, then it cannot be used to match an expression in a query.

The COMPUTE clause is ignored for remote tables.

Any UPDATE statement that attempts to change the value of a computed column fires any triggers associated with the column.

ADD table-constraint clause  Use this clause to add a table constraint. Table constraints place limits on what data columns in the table can hold. When adding or altering table constraints, the optional constraint name allows you to modify or drop individual constraints. Following is a list of the table constraints you can add.

● UNIQUE  Use this subclause to specify that values in the columns specified in column-list must be unique, and, optionally, whether to use a clustered index.

● PRIMARY KEY  Use this subclause to add or alter the primary key for the table, and specify whether to use a clustered index. The table must not already have a primary key that was created by the CREATE TABLE statement or another ALTER TABLE statement.

● foreign-key  Use this subclause to add a foreign key as a constraint. If you use a subclause other than ADD FOREIGN KEY with the ALTER TABLE statement on a table with dependent materialized views, the ALTER TABLE statement fails. For all other clauses, you must disable the dependent materialized views and then re-enable them when your changes are complete.

You can specify a MATCH subclause to control what is considered a match when using a multi-column foreign key. It also allows you to specify uniqueness for the key, thereby eliminating the need to declare uniqueness separately.

ADD PCTFREE clause  Specify the percentage of free space you want to reserve in each table page. The free space is used if rows increase in size when the data is updated. If there is no free space in a table page, every increase in the size of a row on that page requires the row to be split across multiple table
pages, causing row fragmentation and possible performance degradation. A free space percentage of 0 specifies that no free space is to be left on each page—each page is to be fully packed. A high free space percentage causes each row to be inserted into a page by itself. If PCTFREE is not set, or is dropped, the default PCTFREE value is applied according to the database page size (200 bytes for a 4 KB or larger page size). The value for PCTFREE is stored in the ISYSTAB system table. When PCTFREE is set, all subsequent inserts into table pages use the new value, but rows that were already inserted are not affected. The value persists until it is changed. The PCTFREE specification can be used for base, global temporary, or local temporary tables.

**Altering clauses** The following section explains the clauses used for altering the definition for a column or table:

**ALTER column-name column-alteration clause** Use this clause to change attributes for the specified column. If a column is contained in a unique constraint, a foreign key, or a primary key, you can change only the default for the column. However, for any other change, you must delete the key or constraint before the column can be modified.

**column-data-type clause** Use this clause to alter the length or data type of the column. If necessary, the data in the modified column is converted to the new data type. If a conversion error occurs, the operation will fail and the table is left unchanged. You cannot reduce the size of a column. For example, you cannot change a column from a VARCHAR(100) to a VARCHAR(50).

**[ NOT ] NULL clause** Use this clause to change whether NULLs are allowed in the column. If NOT NULL is specified, and the column value is NULL in any of the existing rows, then the operation fails and the table is left unchanged.

**CHECK NULL** Use this clause to delete all check constraints for the column.

**DEFAULT clause** Use this clause to change the default value for the column.

**DEFAULT NULL clause** Use this clause to remove the default value for the column.

**[ CONSTRAINT constraint-name ] CHECK { NULL | ( condition ) } clause** Use this clause to add a CHECK constraint on the column.

If you need to create a CHECK constraint that involves a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

**[ NOT ] COMPRESSED clause** Use this clause to change whether the column is compressed.

**INLINE and PREFIX clauses** The INLINE clause specifies the maximum BLOB size, in bytes, to store within the row. BLOBs smaller than or equal to the value specified by the INLINE clause are stored within the row. BLOBs that exceed the value specified by the INLINE clause are stored outside the row in table extension pages. Also, a copy of some bytes from the beginning of the BLOB may be kept in the row when a BLOB is larger than the INLINE value. Use the PREFIX clause to specify how many bytes are kept in the row. The PREFIX clause can improve the performance of requests that need the prefix bytes of a BLOB to determine if a row is accepted or rejected.

The prefix data for a compressed column is stored uncompressed, so if all the data required to satisfy a request is stored in the prefix, no decompression is necessary.
If neither INLINE nor PREFIX is specified, or if USE DEFAULT is specified, default values are applied as follows:

- For character data type columns, such as CHAR, NCHAR, and LONG VARCHAR, the default value of INLINE is 256, and the default value of PREFIX is 8.

- For binary data type columns, such as BINARY, LONG BINARY, VARBINARY, BIT, VARBIT, LONG VARBIT, BIT VARYING, and UUID, the default value of INLINE is 256, and the default value of PREFIX is 0.

**Note**

It is strongly recommended that you use the default values unless there are specific circumstances that require a different setting. The default values have been chosen to balance performance and disk space requirements. For example, if you set INLINE to a large value, and all the BLOBs are stored inline, row processing performance may degrade. If you set PREFIX too high, you increase the amount of disk space required to store BLOBs since the prefix data is a duplicate of a portion of the BLOB.

If only one of the values is specified, the other value is automatically set to the largest amount that does not conflict with the specified value. Neither the INLINE nor PREFIX value can exceed the database page size. Also, there is a small amount of overhead reserved in a table page that cannot be used to store row data. Therefore, specifying an INLINE value approximate to the database page size can result in a slightly smaller number of bytes being stored inline.

**INDEX and NO INDEX clauses**

When storing BLOBs (character or binary types only), specify INDEX to create BLOB indexes on inserted values that exceed the internal BLOB size threshold (approximately eight database pages). This is the default behavior.

BLOB indexes can improve performance when random access searches within the BLOBs are required. However, for some types of BLOB values, such as images and multimedia files that will never require random-access, performance can improve if BLOB indexing is turned off. To turn off BLOB indexing for a column, specify NO INDEX.

**Note**

A BLOB index is not the same as a table index. A table index is created to index values in one or more columns.

**SET COMPUTE clause**

When a column is created using a COMPUTE clause, its value in any row is the value of the supplied expression. Columns created with this constraint are read-only columns for applications: the value is changed by the database server whenever the row is modified. The COMPUTE expression should not return a non-deterministic value. For example, it should not include a special value such as CURRENT TIMESTAMP, or a non-deterministic function. If a COMPUTE expression returns a non-deterministic value, then it cannot be used to match an expression in a query.

The COMPUTE clause is ignored for remote tables.

Any UPDATE statement that attempts to change the value of a computed column fires any triggers associated with the column.
**ALTER CONSTRAINT** *constraint-name CHECK clause*  Use this clause to alter a named check constraint for the table.

To alter the constraint to specify a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

**Dropping clauses**  The following section explains the DROP clauses:

**DROP DEFAULT**  Drops the default value set for the table or specified column. Existing values do not change.

**DROP COMPUTE**  Removes the COMPUTE attribute for the specified column. This statement does not change any existing values in the table.

**DROP CHECK**  Drops all CHECK constraints for the table or specified column. DELETE CHECK is also accepted.

**DROP CONSTRAINT** *constraint-name*  Drops the named constraint for the table or specified column. DELETE CONSTRAINT is also accepted.

**DROP column-name**  Drops the specified column from the table. DELETE column-name is also accepted. If the column is contained in any index, unique constraint, foreign key, or primary key, then the index, constraint, or key must be deleted before the column can be deleted. This does not delete CHECK constraints that refer to the column.

**DROP UNIQUE (column-name ...)**  Drop the unique constraints on the specified column(s). Any foreign keys referencing this unique constraint are also deleted. DELETE UNIQUE (column-name ...) is also accepted.

**DROP FOREIGN KEY** *fkey-name*  Drop the specified foreign key. DELETE FOREIGN KEY fkey-name is also accepted.

**DROP PRIMARY KEY**  Drop the primary key. All foreign keys referencing the primary key for this table are also deleted. DELETE PRIMARY KEY is also accepted.

**Renaming clauses**  The following section explains the clauses used for renaming parts of a column or table definition:

**RENAME new-table-name**  Change the name of the table to new-table-name. Any applications using the old table name must be modified, as necessary.

**RENAME column-name TO new-column-name**  Change the name of the column to the new-column-name. Any applications using the old column name must be modified, as necessary.

**RENAME CONSTRAINT** *constraint-name TO new-constraint-name*  Change the name of the constraint to the new-constraint-name.

ALTER TABLE...RENAME CONSTRAINT *constraint-name TO new-constraint-name*, when used for an RI constraint, only renames the constraint, not the underlying index or, if applicable, the foreign key role name. To rename the underlying index or the role name, use the ALTER INDEX statement.
**table-alteration clauses**  Use this clause to alter the following table attributes.

**PCTFREE DEFAULT**  Use this clause to change the percent free setting for the table to the default (200 bytes for a 4 KB, and up, page size).

**[ NOT ] ENCRYPTED**  Use this clause to change whether the table is encrypted. To encrypt a table, table encryption must already be enabled on the database. The table is encrypted using the encryption key and algorithm specified at database creation time.

After encrypting a table, any data for that table that was in temporary files or the transaction log before encryption still exists in unencrypted form. To address this, restart the database to remove the temporary files. Run the Backup utility (dbbackup) with the -o option, or use the BACKUP statement, to back up the transaction log and start a new one.

When table encryption is enabled, table pages for the encrypted table, associated index pages, temporary file pages, and transaction log pages containing transactions on encrypted tables are encrypted.

**DISABLE VIEW DEPENDENCIES clause**  Use this clause to disable dependent regular views.
Dependent materialized views are not disabled; you must disable each dependent materialized view by executing an ALTER MATERIALIZED VIEW...DISABLE statement.

**ALTER OWNER clause**  To alter the owner of a table:

○ You must have ALTER ANY OBJECT OWNER privilege.

○ You must have one of the following privileges: ALTER privilege on the table, ALTER ANY TABLE privilege, or ALTER ANY OBJECT privilege.

○ The new owner cannot already own a table with the same name.

○ Enabled materialized views cannot reference the table.

**PRESERVE or DROP PRIVILEGES**  If you do not want the new owner to have the same privileges as the old owner, you can specify DROP PRIVILEGES to drop all explicitly-granted privileges that allow a user access to the table. Implicitly-granted privileges given to the owner of the table are given to the new owner and dropped from the old owner.

**PRESERVE or DROP FOREIGN KEYS**  To prevent the new owner from accessing data in referenced tables, you can specify DROP FOREIGN KEYS to drop all foreign keys within the table, as well as all foreign keys referring to the table.

**Remarks**

The ALTER TABLE statement changes table attributes (column definitions, constraints, and so on) in an existing table.

The database server keeps track of object dependencies in the database. Alterations to the schema of a table may impact dependent views. Also, if there are materialized views that are dependent on the table you are attempting to alter, you must first disable them using the ALTER MATERIALIZED VIEW...DISABLE statement.

You cannot use ALTER TABLE on a local temporary table.
ALTER TABLE is prevented whenever the statement affects a table that is currently being used by another connection. ALTER TABLE can be time-consuming, and the database server does not process other requests referencing the table while the statement is being processed.

If you alter a column that a text index defined as IMMEDIATE REFRESH is built on, the text index is immediately rebuilt. If the text index is defined as AUTO REFRESH or MANUAL REFRESH, the text index is rebuilt the next time it is refreshed.

When you execute an ALTER TABLE statement, the database server attempts to restore column privileges on dependent views that are automatically recompiled. Privileges on columns that no longer exist in the recompiled views are lost.

ALTER TABLE requires exclusive access to the table.

Global temporary tables cannot be altered unless all users that have referenced the temporary table have disconnected.

This statement cannot be used within a snapshot transaction.

**Privileges**

You must be the owner of the table, or have one of the following privileges:

- ALTER privilege on the table
- ALTER ANY TABLE system privilege
- ALTER ANY OBJECT system privilege
- MANAGE ANY DBSPACE system privilege

To alter the table owner, you must also have the ALTER ANY OBJECT OWNER system privilege.

To create, alter, or drop indexes on a table, you must have the CREATE ANY INDEX, ALTER ANY INDEX, or DROP ANY INDEX system privilege, respectively.

**Side effects**

Automatic commit.

A checkpoint is carried out at the beginning of the ALTER TABLE operation, and further checkpoints are suspended until the ALTER operation completes.

Once you alter a column or table, any stored procedures, views, or other items that refer to the altered column may no longer work.

If you change the declared length or type of a column, or drop a column, the statistics for that column are dropped.
See also

- “Events” [SQL Anywhere Server - Database Administration]
- “sa_reset_identity system procedure” on page 1214
- “Choosing between sequences and AUTOINCREMENT values” [SQL Anywhere Server - SQL Usage]
- “Reloading tables with AUTOINCREMENT columns” [SQL Anywhere 16 - Changes and Upgrading]
- “ALTER MATERIALIZED VIEW statement” on page 450
- “BACKUP statement” on page 516
- “ALTER INDEX statement” on page 441
- “CREATE TABLE statement” on page 690
- “DROP TABLE statement” on page 780
- “MATCH clause, CREATE TABLE statement” on page 699
- “Backup utility (dbbackup)” [SQL Anywhere Server - Database Administration]
- “SQL data types” on page 89
- “Table alteration” [SQL Anywhere Server - SQL Usage]
- “The special IDENTITY column” [SQL Anywhere Server - SQL Usage]
- “Special values” on page 65
- “Table and column constraints” [SQL Anywhere Server - SQL Usage]
- “Use of a sequence to generate unique values” [SQL Anywhere Server - SQL Usage]
- “allow_nulls_by_default option” [SQL Anywhere Server - Database Administration]
- “Encrypting a table” [SQL Anywhere Server - Database Administration]
- “Clustered indexes” [SQL Anywhere Server - SQL Usage]
- “View dependencies” [SQL Anywhere Server - SQL Usage]
- “Tip: Update column statistics” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- **SQL/2008**  
  ALTER TABLE is a core feature. In the SQL/2008 standard, ADD COLUMN and DROP COLUMN are supported as core features, as are ADD CONSTRAINT and DROP CONSTRAINT. ALTER [COLUMN] is SQL feature F381, as is the ability to add, modify, or drop a DEFAULT value for a column. In SQL/2008, altering the data type of a column is performed by specifying the SET DATA TYPE clause, which is SQL language feature F382. Conversely, SQL Anywhere supports modifying a column's data type through the ALTER clause directly.

  Other clauses supported by SQL Anywhere, including ALTER CONSTRAINT, RENAME, PCTFREE, ENCRYPTED, and DISABLE MATERIALIZED VIEW, are vendor extensions. Support for extensions to column definitions, and column and table constraint definitions, are vendor extensions to SQL/2008 or are specific optional features of SQL/2008.

- **Transact-SQL**  
  ALTER TABLE is supported by Adaptive Server Enterprise. Adaptive Server Enterprise supports the ADD COLUMN and DROP COLUMN clauses, in addition to ADD CONSTRAINT and DROP CONSTRAINT. Adaptive Server Enterprise uses MODIFY rather than the keyword ALTER for the ALTER clause. Adaptive Server Enterprise uses the REPLACE clause for altering a column's DEFAULT value. In Adaptive Server Enterprise, ALTER TABLE is also used to enable/disable triggers for a specific table, a feature that is not supported in SQL Anywhere.
**Example**

The following example adds a new timestamp column, TimeStamp, to the Customers table. To run this next statement, you must have ALTER privilege on the Customers table.

```sql
ALTER TABLE GROUPO.Customers
ADD TimeStamp AS TIMESTAMP DEFAULT TIMESTAMP;
```

The following example drops the new timestamp column, TimeStamp that you added in the previous example.

```sql
ALTER TABLE GROUPO.Customers
DROP TimeStamp;
```

The Street column in the Customers table can currently hold up to 35 characters. To allow it to hold up to 50 characters, execute the following:

```sql
ALTER TABLE GROUPO.Customers
ALTER Street CHAR(50);
```

The following example adds a column to the Customers table, assigning each customer a sales contact. To run this next statement, you must also have the CREATE ANY INDEX system privilege because it creates a foreign key.

```sql
ALTER TABLE GROUPO.Customers
ADD SalesContact INTEGER
REFERENCES GROUPO.Employees (EmployeeID)
ON UPDATE CASCADE
ON DELETE SET NULL;
```

This foreign key is constructed with cascading updates and is set to NULL on deletes. If an employee has their employee ID changed, the column is updated to reflect this change. If an employee leaves the company and has their employee ID deleted, the column is set to NULL.

The following example creates a foreign key, FK_SalesRepresentative_EmployeeID2, on the SalesOrders.SalesRepresentative column, linking it to Employees.EmployeeID. You must have ALTER privilege on the SalesOrder table to execute the following statement:

```sql
ALTER TABLE GROUPO.SalesOrders
ADD CONSTRAINT FK_SalesRepresentative_EmployeeID2
FOREIGN KEY (SalesRepresentative)
REFERENCES GROUPO.Employees (EmployeeID);
```

The following example adds a column where the default is AUTOINCREMENT. In this example, all existing customer rows are modified to have a nullable AUTOINCREMENT column with the column value assigned, but the database server does not guarantee which row is assigned which value:

```sql
ALTER TABLE GROUPO.Customers
ADD Surrogate_key INTEGER DEFAULT AUTOINCREMENT;
```

The following example changes the owner of the fictitious table mytable to Bob:

```sql
ALTER TABLE mytable ALTER OWNER TO bob;
```
ALTER TEXT CONFIGURATION statement

Alters a text configuration object.

Syntax

```
ALTER TEXT CONFIGURATION [ owner.]config-name
STOPLIST stoplist-string
| DROP STOPLIST
|   { MINIMUM | MAXIMUM } TERM LENGTH integer }
| TERM BREAKER { GENERIC [ EXTERNAL NAME external-call ] | NGRAM }
| PREFILTER EXTERNAL NAME external-call
| DROP PREFILTER
| SAVE OPTION VALUES [ FROM CONNECTION ]
```

```
external-call : "[ operating-system: ]library-function-name@library-name[;...]
```

```
operating-system : UNIX
```

Parameters

**STOPLIST clause**  Use this clause to create or replace the list of terms to ignore when building a text index. Using this text configuration object, terms specified in this list are also ignored in a query. Separate stoplist terms with spaces. For example, `STOPLIST 'because about therefore only'`. Stoplist terms cannot contain whitespace.

Samples of stoplists for different languages are located in `%SQLANYSAMP16\SQLAnywhere\SQL`.

Stoplist terms should not contain non-alphanumeric characters. The stoplist length must be less than 8000 bytes.

Carefully consider whether you want to put terms in your stoplist.

**DROP STOPLIST clause**  Use this clause to drop the stoplist for a text configuration object.

**MINIMUM TERM LENGTH clause**  The value specified in the MINIMUM TERM LENGTH clause is ignored when using NGRAM text indexes.

The minimum length, in characters, of a term to include in the text index. Terms that are shorter than this setting are ignored when building or refreshing the text index. The value of this option must be greater than 0. If you set this option to be higher than MAXIMUM TERM LENGTH, the value of MAXIMUM TERM LENGTH is automatically adjusted to be the same as the new MINIMUM TERM LENGTH value.

**MAXIMUM TERM LENGTH clause**  With NGRAM text indexes, use the MAXIMUM TERM LENGTH clause to set the size of the n-grams into which strings are broken.

With GENERIC text indexes, use the MAXIMUM TERM LENGTH clause to set the maximum length, in characters, of a term to include in the text index. Terms that are longer than this setting are ignored when building or refreshing the text index. The value of MAXIMUM TERM LENGTH must be less than or equal to 60. If you set this option to be lower than MINIMUM TERM LENGTH, the value of MINIMUM TERM LENGTH is automatically adjusted to be the same as the new MAXIMUM TERM LENGTH value.
**TERM BREAKER clause**  The name of the algorithm to use for separating column values into terms. The choices are GENERIC (the default) or NGRAM.

- **GENERIC**  For GENERIC, you can use the built-in GENERIC term breaker algorithm by specifying TERM BREAKER GENERIC, or you can specify an external algorithm using the TERM BREAKER GENERIC EXTERNAL NAME clause.

  The built-in GENERIC algorithm treats any string of one or more alphanumerics, separated by non-alphanumerics, as a term.

  Specify the TERM BREAKER GENERIC EXTERNAL NAME clause to specify an entry point to a term breaker function in an external library. This is useful if you have custom requirements for how you want terms broken up before they are indexed or queried (for example, if you want an apostrophe to be considered as part of a term and not as a term breaker).

  external-call can specify more than one function and/or library, and can include the file extension of the library, which is typically .dll on Windows, and .so on Unix. In the absence of the file extension, the database server defaults to the platform-specific file extension for libraries. For example, EXTERNAL NAME 'TermBreakFunct1@myTBlib;Unix:TermBreakFunct2@myTBlib' calls the TermBreakFunct1 function from myTBlib.dll on Windows, and the TermBreakFunct2 function from myTBlib.so on Unix.

- **NGRAM**  The built-in NGRAM algorithm breaks strings into n-grams. An n-gram is an n-character substring of a larger string. The NGRAM term breaker is required for fuzzy (approximate) matching, or for documents that do not use whitespace or non-alphanumeric characters to separate terms, if no external term breaker is specified.

**PREFILTER EXTERNAL NAME clause**  Specify the PREFILTER EXTERNAL NAME clause to specify an entry point to a prefilter function in an external library. This is useful if text data needs to be extracted from binary data (for example, PDF). It is also useful if the text you want to index contains formatting information and/or images that you want to strip out before indexing the data (for example, HTML).

  external-call can specify more than one function and/or library, and can include the file extension of the library, which is typically .dll on Windows, and .so on Unix. In the absence of the file extension, the database server defaults to the platform-specific file extension for libraries. For example, PREFILTER EXTERNAL NAME 'PrefilterFunct1@myPreFilterlib;Unix:PrefilterFunct2@myPreFilterlib' calls the PrefilterFunct1 function from myPreFilterlib.dll on Windows, and the PrefilterFunct2 function from myPreFilterlib.so on Unix.

**DROP PREFILTER clause**  Use the DROP PREFILTER clause to drop use of the specified prefiltering library for the text configuration object. Prefiltering is no longer performed when the database server builds indexes that use this text configuration object.

**SAVE OPTION VALUES clause**  When a text configuration object is created, the current date_format, time_format, timestamp_format, and timestamp_with_time_zone_format database options reflect how DATE, TIME, and TIMESTAMP columns are saved with the text configuration object. Use the SAVE OPTION VALUES clause to update the option values saved for the text configuration object to reflect the options currently in effect for the connection.
Remarks

Before changing the term length settings, read about the impact of various settings on what gets indexed and how query terms are interpreted.

Text indexes are dependent on a text configuration object. Before using this statement you must truncate dependent AUTO or MANUAL REFRESH text indexes, and drop any IMMEDIATE REFRESH text indexes.

To view the settings for text configuration objects, query the SYSTEXTCONFIG system view.

Privileges

You must be the owner of the text configuration object, or have one of the following privileges:

- ALTER privilege on the text configuration object
- ALTER ANY TEXT CONFIGURATION system privilege
- ALTER ANY OBJECT system privilege

If you are altering an external term breaker, you must also have the CREATE EXTERNAL REFERENCE system privilege.

When specifying or dropping a prefilter, or specifying an external term breaker, you must also have the ALTER ANY TEXT CONFIGURATION or ALTER ANY OBJECT system privilege.

Side effects

Automatic commit

See also

- “Altering a text configuration object” [SQL Anywhere Server - SQL Usage]
- “Viewing text index terms and settings (Sybase Central)” [SQL Anywhere Server - SQL Usage]
- “What to specify when creating or altering text configuration objects” [SQL Anywhere Server - SQL Usage]
- “Tutorial: Performing a full text search on a GENERIC text index” [SQL Anywhere Server - SQL Usage]
- “Tutorial: Performing a fuzzy full text search” [SQL Anywhere Server - SQL Usage]
- “CREATE TEXT CONFIGURATION statement” on page 709
- “DROP TEXT CONFIGURATION statement” on page 781
- “sa_char_terms system procedure” on page 1095
- “sa_nchar_terms system procedure” on page 1195
- “sa_refresh_text_indexes system procedure” on page 1211
- “sa_text_index_stats system procedure” on page 1256
- “SYSTEXTCONFIG system view” on page 1405

Standards and compatibility

- SQL/2008 Vendor extension.
Example

The following statements create a text configuration object, myTextConfig, and then change the maximum term length to 16. To run this statement, you’ll need the CREATE TEXT CONFIGURATION system privilege:

```sql
CREATE TEXT CONFIGURATION myTextConfig FROM default_char;
ALTER TEXT CONFIGURATION maxTerm16
  MAXIMUM TERM LENGTH 16;
```

The following statement adds a stoplist to the myTextConfig configuration object:

```sql
ALTER TEXT CONFIGURATION myTextConfig
  STOPLIST 'because about therefore only';
```

The following statement configures an external term breaker for the myTextConfig text configuration object. Both the Windows and Unix interfaces are specified.

```sql
ALTER TEXT CONFIGURATION myTextConfig
  TERM BREAKER GENERIC
  EXTERNAL NAME
  'my_termbreaker@termbreaker.dll;Unix:my_termbreaker@libtermbreaker_r.so'
```

The following example configures an external prefilter for the myTextConfig text configuration object. Both the Windows and Unix interfaces are specified.

```sql
ALTER TEXT CONFIGURATION myTextConfig
  PREFILTER EXTERNAL NAME
  'html_xml_filter@html_xml_filter.dll;UNIX:html_xml_filter@libhtml_xml_filter_r.so';
```

The following example drops the external prefilter for the myTextConfig text configuration object.

```sql
ALTER TEXT CONFIGURATION myTextConfig DROP PREFILTER;
```

ALTERN TEXT INDEX statement

Alters the definition of a text index.

Syntax

```sql
ALTER TEXT INDEX [ owner.]text-index-name
  ON [ owner.]table-name
  alter-clause
alter-clause :
  rename-object
  refresh-alteration
rename-object :
  RENAME { AS | TO } new-name
refresh-alteration :
  { MANUAL REFRESH
  | AUTO REFRESH [ EVERY integer { MINUTES | HOURS } ] }
```
Parameters

**RENAME clause**  Use the RENAME clause to rename the text index.

**REFRESH clause**  Specify the REFRESH clause to set the refresh type for the text index.

Remarks

Once a text index is created, you cannot change it to, or from, IMMEDIATE REFRESH. If either of these changes is required, you must drop and recreate the text index.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

You can only alter a text index built on a materialized view by renaming it. You cannot change the refresh type for a text index built on a materialized view.

Privileges

To alter a text index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

To alter a text index on a materialized view, you must be the owner of the materialized, or have one of the following privileges:

- ALTER ANY INDEX system privilege
- ALTER ANY OBJECT system privilege

Side effects

Automatic commit

See also

- “CREATE TEXT INDEX statement” on page 711
- “ALTER TEXT INDEX statement” on page 504
- “DROP TEXT INDEX statement” on page 782
- “REFRESH TEXT INDEX statement” on page 928
- “TRUNCATE TEXT INDEX statement” on page 1021
- “COMMENT statement” on page 538
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]
- “Viewing text index terms and settings (Sybase Central)” [SQL Anywhere Server - SQL Usage]
- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Tutorial: Performing a full text search on a GENERIC text index” [SQL Anywhere Server - SQL Usage]
- “Tutorial: Performing a fuzzy full text search” [SQL Anywhere Server - SQL Usage]
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The first statement creates a text index, txt_index_manual, defining it as MANUAL REFRESH. The second statement alters the text index to refresh automatically every day. The third statement renames the text index to txt_index_daily.

```
CREATE TEXT INDEX txt_index_manual ON GROUPO.MarketingInformation
  ( Description )
  MANUAL REFRESH;
ALTER TEXT INDEX txt_index_manual ON GROUPO.MarketingInformation
  AUTO REFRESH EVERY 24 HOURS;
ALTER TEXT INDEX txt_index_manual ON GROUPO.MarketingInformation
  RENAME AS txt_index_daily;
```

ALTER TRACE EVENT SESSION statement

Adds or removes trace events from a session, adds or removes targets from a session, or starts or stops a trace session.

Syntax

```
ALTER TRACE EVENT SESSION session-name
  { add-drop-trace-event [...]
      | add-drop-target [...]
    | STATE = { START | STOP }
  }

add-drop-trace-event :
  { ADD TRACE EVENT trace-event-name ... | DROP TRACE EVENT trace-event-name }

add-drop-target :
  { ADD TARGET target-name ... | DROP TARGET target-name }

target-name : filename
```

Parameters

- **ADD TRACE EVENT**   The name of the trace event being added to the session.
- **DROP TRACE EVENT**   The name of the trace event being removed from the session.
- **ADD TARGET**        The name of the target being added to the session. Information about the trace events that are part of the session is logged to this target.
- **DROP TARGET**       The name of the target being removed from the session.

Remarks

Adding or dropping trace events or targets from a running trace session causes the session to pause briefly to make the changes and then resume after the changes are made. You do not need to stop a tracing session to add or drop trace events or targets. Adding or removing a trace event from a session that has already started has the side effect that some trace events are missed while a session is temporarily paused.
System privileges
You must have the MANAGE ANY TRACE SESSION system privilege.

Side effects
None

See also
- “Event tracing” [SQL Anywhere Server - Database Administration]
- “CREATE TEMPORARY TRACE EVENT statement” on page 705
- “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
- “DROP TRACE EVENT statement” on page 783
- “DROP TRACE EVENT SESSION statement” on page 784
- “NOTIFY TRACE EVENT statement” on page 903
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_fields system procedure” on page 1312
- “sp_trace_event_sessions system procedure” on page 1317
- “sp_trace_event_session_events system procedure” on page 1314
- “sp_trace_event_session_targets system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example creates a trace event called my_event, creates a trace event session called my_session, and starts the session:

```sql
CREATE TEMPORARY TRACE EVENT my_event( id INTEGER, information LONG VARCHAR );
CREATE TEMPORARY TRACE EVENT SESSION my_session
  ADD TRACE EVENT my_event, -- user event
  ADD TRACE EVENT SYS_ConsoleLog__Information -- system event
  ADD TARGET FILE ( SET filename_prefix='my_trace_file' ); -- add a target
ALTER TRACE EVENT SESSION my_session
  STATE = START;
```

ALTER TRIGGER statement
Replaces a trigger definition with a modified version. You must include the entire new trigger definition in the ALTER TRIGGER statement.

Syntax 1 - Change the definition of a trigger

```
ALTER TRIGGER trigger-name trigger-definition

trigger-definition : CREATE TRIGGER trigger syntax
```
Syntax 2 - Obfuscate a trigger definition

ALTER TRIGGER trigger-name ON [owner.] table-name SET HIDDEN

Remarks

- **Syntax 1**  The ALTER TRIGGER statement is identical in syntax to the CREATE TRIGGER statement except for the first word.

  Either the Transact-SQL or Watcom SQL form of the CREATE TRIGGER syntax can be used.

- **Syntax 2**  You can use SET HIDDEN to obfuscate the definition of the associated trigger and cause it to become unreadable. The trigger can be unloaded and reloaded into other databases. If SET HIDDEN is used, debugging using the debugger does not show the trigger definition, nor is it available through procedure profiling.

  **Note**
  The SET HIDDEN operation is irreversible.

Privileges

You must be the owner of the underlying table, or have one of the following privileges:

- ALTER privilege on the underlying table with the CREATE ANY OBJECT system privilege
- ALTER ANY TRIGGER system privilege
- ALTER ANY OBJECT system privilege

To alter a trigger on a view owned by someone else, you must have either the ALTER ANY TRIGGER and ALTER ANY VIEW system privileges, or you must have the ALTER ANY OBJECT system privilege.

Side effects

Automatic commit.

See also

- “CREATE TRIGGER statement” on page 713
- “CREATE TRIGGER statement [T-SQL]” on page 720
- “DROP TRIGGER statement” on page 785
- “Hiding the contents of a procedure, function, trigger, event, or view” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008  Vendor extension.

**ALTER USER statement**

Alters user settings.
Syntax 1 - Change the definition of a database user

ALTER USER user-name
[ IDENTIFIED BY password ]
[ LOGIN POLICY policy-name ]
[ FORCE PASSWORD CHANGE { ON | OFF } ]

Syntax 2 - Revert a user's login policy to the original values

ALTER USER user-name
[ RESET LOGIN POLICY ]

Syntax 3 - Refresh the Distinguished Name (DN) for an LDAP user

ALTER USER user-name
REFRESH DN

Syntax 4 - Change a password part

ALTER USER user-name
[ IDENTIFIED { FIRST | LAST } BY password-part ]

Parameters

user-name The name of the user.

IDENTIFIED clause The password of the user. A user without a password cannot connect to the database.

○ IDENTIFIED BY clause Use this clause to reset the password for a user.

○ IDENTIFIED { FIRST | LAST } BY clause Use this clause to reset part of the password for a user who has a dual control password. A user with dual control password has the CHANGE_PASSWORD_DUAL_CONTROL login policy option enabled for their login policy.

Two administrators are required to reset a dual control password. One administrator executes the IDENTIFIED FIRST BY clause to set the first part of the password and another administrator executes the IDENTIFIED LAST BY clause to set the last part of the password. The user combines the two password parts and uses this combined password to connect to the database.

See “Dual control passwords” [SQL Anywhere Server - Database Administration].

policy-name The name of the login policy to assign the user. No change is made if the LOGIN POLICY clause is not specified.

FORCE PASSWORD CHANGE clause Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in the user's policy.

RESET LOGIN POLICY clause Reverts the settings of a user's login policy to the original values. When you reset a login policy, a user can access an account that has been locked for exceeding a login policy option limit such as max_failed_login_attempts or max_days_since_login.

REFRESH DN clause REFRESH DN clears the Distinguished Name (DN) and timestamp of the user so that at the time of the next LDAP authentication, the search for the DN is done. If the authentication
succeeds during the next LDAP authentication of this user then both the DN and the timestamp are updated with the new DN and current time.

Remarks
The following list describes the requirements for user IDs and passwords. The requirements and restrictions for a password part are the same as those described for a password except that the maximum length of each part is 127 bytes.

- User IDs cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons

- Passwords are case-sensitive and they cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons
  - be longer than 255 bytes in length

The verify_password_function login policy option can be used to specify a function to implement password rules (for example, passwords must include at least one digit). If a password verification function is used, you cannot specify more than one user ID and password in the GRANT CONNECT statement.

If you set the password_expiry_on_next_login value to ON, the user's password expires immediately when they next login even if they are assigned to the same policy. You can use the ALTER USER and LOGIN POLICY clauses to force a user to change their password when they next login.

The ALTER USER...REFRESH DN syntax clears the Distinguished Name (DN) and timestamp of a user so that during the next LDAP authentication, a search for the DN is performed, instead of using the cached DN, which can become out of date. If the authentication succeeds, then both the DN and the timestamp are updated with the new DN and current time.

Privileges
Any user can change their own password.

To change passwords for other users, you must have the CHANGE PASSWORD system privilege.

For all other changes to other users, including forcing users to change their password, you must have the MANAGE ANY USER system privilege.

Side effects
None.

Standards and compatibility
- SQL/2008 Vendor extension.
Example

The following statement alters a user named SQLTester, setting their password to welcome123, setting their login policy to Test1, and allowing them to bypass a forced password change.

```
ALTER USER SQLTester IDENTIFIED BY welcome123
LOGIN POLICY Test1
FORCE PASSWORD CHANGE off;
```

The following example refreshes the LDAP Distinguished Name for user myusername.

```
ALTER USER myusername REFRESH DN;
```

See also

- “ALTER LOGIN POLICY statement” on page 445
- “COMMENT statement” on page 538
- “CREATE LOGIN POLICY statement” on page 607
- “CREATE USER statement” on page 721
- “DROP LOGIN POLICY statement” on page 761
- “DROP USER statement” on page 786
- “verify_password_function option” [SQL Anywhere Server - Database Administration]
- “Login policies” [SQL Anywhere Server - Database Administration]
- “Assigning a login policy to an existing user” [SQL Anywhere Server - Database Administration]
- “GRANT CONNECT statement” on page 833

**ALTER VIEW statement**

Replaces a view definition with a modified version.

**Syntax 1 - Change the definition of a view**

```
ALTER VIEW
    [ owner.]
view-name
    [ ( column-name, ... ) ]
AS
query-expression
    [ WITH CHECK OPTION ]
```

**Syntax 2 - Change the attributes of a view**

```
ALTER VIEW
    [ owner.]
view-name
    { SET HIDDEN | RECOMPILE | DISABLE | ENABLE }
```

**Parameters**

- **AS clause**  The SELECT statement on which the view is based. The SELECT statement must not refer to local temporary tables. Also, `query-expression` can have a GROUP BY, HAVING, WINDOW, or ORDER BY clause, and can contain UNION, EXCEPT, INTERSECT, or a common table expression.

Query semantics dictate that the order of the rows returned is undefined unless the query combines an ORDER BY clause with a TOP or FIRST clause in the SELECT statement.

- **WITH CHECK OPTION clause**  The WITH CHECK OPTION clause rejects any updates and inserts to the view that do not meet the criteria of the view as defined by its `query-expression`. 
**SET HIDDEN clause**  Use the SET HIDDEN clause to obfuscate the definition of the view and cause the view to become hidden from view, for example in Sybase Central. Explicit references to the view still work.

**Note**
The SET HIDDEN operation is irreversible.

**RECOMPILE clause**  Use the RECOMPILE clause to re-create the column definitions for the view. This clause is identical in functionality to the ENABLE clause, except that it can be used on a view that is not disabled. When a view is recompiled, the database server restores the column privileges based on the column names specified in the new view definition. The existing privileges are lost when a column no longer exists after the recompilation.

**DISABLE clause**  Use the DISABLE clause to disable the view from use by the database server.

**ENABLE clause**  Use the ENABLE clause to enable a disabled view. Enabling the view causes the database server to re-create the column definitions for the view. Before you enable a view, you must enable any views upon which it depends.

**Remarks**

The *query-expression* can specify a TOP n, FIRST, or LIMIT clause even if there is no ORDER BY clause. At most the specified number of rows are returned, but the order of the rows returned is not defined, so an ORDER BY could be specified but it is not required. When a view is used in a query, the ORDER BY in the view does not determine the order of rows in the query, even if there are no other tables in the FROM clause. That means that an ORDER BY should only be included in a view if it is needed to select which rows are included by a TOP n, FIRST, or LIMIT clause. Otherwise, the ORDER BY clause has no effect and it is ignored by the database server.

If you execute an ALTER VIEW statement on a view that has one or more INSTEAD OF triggers, an error is returned. You must drop the trigger before the view can be dropped or altered.

If you alter a view owned by another user, you must qualify the name by including the owner (for example, GROUPO.ViewSalesOrders). If you don't qualify the name, the database server looks for a view with that name owned by you and alters it. If there isn't one, it returns an error.

When you alter a view, existing privileges on the view are maintained, and do not have to be reassigned. Instead of using the ALTER VIEW statement, you could also drop the view and recreate it using the DROP VIEW and CREATE VIEW, respectively. However, if you do so, privileges on the view need to be reassigned.

A query can specify a TOP n, FIRST, or LIMIT clause even if there is no ORDER BY clause. At most the specified number of rows are returned, but the order of the rows returned is not defined, so an ORDER BY could be specified but it is not required. When a view is used in a query, the ORDER BY in the view does not determine the order of rows in the query, even if there are no other tables in the FROM clause. That means that an ORDER BY should only be included in a view if it is needed to select which rows are included by a TOP n, FIRST, or LIMIT clause. Otherwise, the ORDER BY clause has no effect and it is ignored by the database server.
After completing the view alteration using Syntax 1, the database server recompiles the view. Depending on the type of change you made, if there are dependent views, the database server attempts to recompile them as well. If you have made a change that impacts a dependent view, you may need to alter the definition for the dependent view as well.

**Caution**
If the SELECT statement defining the view contained an asterisk (*), the number of the columns in the view may change if columns have been added or deleted from the underlying tables. The names and data types of the view columns may also change.

- **Syntax 1**  This syntax is used to alter the structure of the view. Unlike altering tables where your change may be limited to individual columns, altering the structure of a view requires you to replace the entire view definition with a new definition, much as you would for creating the view.

- **Syntax 2**  This syntax is used to change attributes for the view, such as whether the view definition is hidden.

When you use SET HIDDEN, the view can be unloaded and reloaded into other databases. If SET HIDDEN is used, debugging using the debugger does not show the view definition, nor is it be available through procedure profiling. If you need to change the definition of a hidden view, you must drop the view and create it again using the CREATE VIEW statement.

When you use the DISABLE clause, the view is no longer available for use by the database server for answering queries. Disabling a view is similar to dropping it, except that the view definition remains in the database. Disabling a view also disables any dependent views. Therefore, the DISABLE clause requires exclusive access not only to the view being disabled, but also any dependent views, since they are disabled too.

**Privileges**
You must be the owner of the view, or have one of the following privileges:

- ALTER privilege on the view
- ALTER ANY VIEW system privilege
- ALTER ANY OBJECT system privilege

You must also have permission to select from the underlying objects for the view.

**Side effects**
Automatic commit.

All procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the new view definition. The unloading and loading of procedures and triggers can have a performance impact if you are regularly altering views.
ATTACH TRACING statement

Starts a diagnostic tracing session (starts sending diagnostic information to the diagnostic tables).

Syntax

```
ATTACH TRACING TO { LOCAL DATABASE | connect-string }
[ LIMIT { size | history } ]
```

- **connect-string**: the connection string for the database
- **size**: `SIZE nnn { MB | GB }`
- **history**: `HISTORY nnn { MINUTES | HOURS | DAYS }
```

- **nnn**: integer

Parameters

- **connect-string**  The connection string is required to connect to the database receiving the tracing information. This parameter is only required when the database being profiled is different from the database receiving the data.

  The following connection parameters are allowed in `connect-string`: DBF, DBKEY, DBN, Host, Server, LINKS, PWD, UID.

  Specify the DBF relative to the database server that you want to connect to. If you do not specify a different database server, then the database server to which you are currently connected attempts to start the tracing database identified by the DBF connection parameter.

  An error is returned if you specify the DBF parameter with the LINKS or Server connection parameters.

- **LIMIT clause**  The volume limit of data stored in the tracing database, either by size, or by length of time.
Remarks
The ATTACH TRACING statement is used to start a tracing session for the database you want to profile. You can only use it once a tracing level has been set. You can set the tracing level using Sybase Central, or using the sa_set_tracing_level system procedure.

Once a session is started, tracing information is generated according to the tracing levels set in the sa_diagnostic_tracing_level table. You can send the tracing data to tracing tables within the same database that is being profiled by specifying LOCAL DATABASE. Alternatively, you can send the tracing data to a separate tracing database by specifying a connection string (connect-string) to that database. The tracing database must already exist, and you must have permissions to access it.

You can limit the amount of tracing data to store using the LIMIT SIZE or LIMIT HISTORY clauses. Use the LIMIT SIZE clause when you want to limit the volume of tracing data to a certain size, as measured in megabytes or gigabytes. Use the LIMIT HISTORY clause to limit the volume of tracing data to a period of time, as measured in minutes, hours, or days. For example, HISTORY 8 DAYS limits the amount of tracing data stored in the tracing database to 8 days' worth.

To start a tracing session, TCP/IP must be running on the database server(s) on which the tracing database and production database are running.

Packets that contain potentially sensitive data are visible on the network interface, even when tracing to a local database. For security purposes, you can specify encryption in the connection string.

To see the current tracing levels set for a database, look in the sa_diagnostic_tracing_level table.

To see where tracing data is being sent to, examine the SendingTracingTo database property.

You must be connected to the database being profiled to execute this statement.

Privileges
You must have the DIAGNOSTICS system role and the MANAGE PROFILING system privilege.

Side effects
None.

See also
- “DETACH TRACING statement” on page 750
- “REFRESH TRACING LEVEL statement” on page 930
- “-x database server option” [SQL Anywhere Server - Database Administration]
- “sa_diagnostic_tracing_level table” on page 1081
- “sa_set_tracing_level system procedure” on page 1245
- “SendingTracingTo database property” [SQL Anywhere Server - Database Administration]
- “Connection parameters” [SQL Anywhere Server - Database Administration]
- “TCP/IP protocol” [SQL Anywhere Server - Database Administration]
- “Diagnostic tracing” [SQL Anywhere Server - SQL Usage]
Standards and compatibility
  - SQL/2008  Vendor extension.

Examples
The following example sets the tracing level to 1 using the sa_set_tracing_level system procedure. Then it
starts a tracing session. Tracing data generated for the local database will be sent to the mytracingdb
tracing database on another computer, as shown by the specified connection string. A maximum of two
hours of tracing data will be maintained during the tracing session. The ATTACH TRACING statement
example is all on one line.

  CALL sa_set_tracing_level( 1 );
  ATTACH TRACING TO
    'uid=DBA;pwd=sql;server=remotedbsrv;dbn=mytracingdb;host=winxp-32'
    LIMIT HISTORY 2 HOURS;

BACKUP statement
Backs up a database and transaction log.

Syntax 1 - Image backup
  BACKUP DATABASE
    DIRECTORY backup-directory
      [ backup-option [ backup-option ... ] ]

  backup-directory : { string | variable }

  backup-option :
    WAIT BEFORE START
    | WAIT AFTER END
    | DBFILE ONLY
    | TRANSACTION LOG ONLY
    | TRANSACTION LOG RENAME [ MATCH ]
    | TRANSACTION LOG TRUNCATE
    | ON EXISTING ERROR
    | WITH COMMENT comment string
    | HISTORY { ON | OFF }
    | AUTO TUNE WRITERS { ON | OFF }
    | WITH CHECKPOINT LOG { AUTO | COPY | NO COPY | RECOVER }

Syntax 2 - Archive backup
  BACKUP DATABASE TO archive-root
    [ backup-option [ backup-option ... ] ]

  archive-root : { string | variable }

  backup-option :
    WAIT BEFORE START
    | WAIT AFTER END
    | DBFILE ONLY
    | TRANSACTION LOG ONLY
    | TRANSACTION LOG RENAME [ MATCH ]
    | TRANSACTION LOG TRUNCATE
Parameters

DIRECTORY clause The target location on disk for the backup files, relative to the database server's current directory at startup. If the directory does not exist, it is created. Specifying an empty string as a directory allows you to rename or truncate the transaction log without first making a copy of it. Do not use this clause if you are using database mirroring.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

WAIT BEFORE START clause This clause delays the backup until there are no active transactions. All other activity on the database is prevented and a checkpoint is performed.

Using this clause with the WITH CHECKPOINT LOG NO COPY clause verifies that the backup copy of the database does not require recovery and allows you to start the backup copy of the database in read-only mode and validate it. When you validate the backup database, you do not need to make an additional copy of the database.

WAIT AFTER END clause This clause ensures that all transactions are completed before the transaction log is renamed or truncated. The database server waits for other connections to commit or rollback any open transactions before finishing the backup. Use this clause with caution as new, incoming transactions can cause the backup to wait indefinitely.

DBFILE ONLY clause This clause makes backup copies of the main database file and all associated dbspaces, but not the transaction log. You cannot use the DBFILE ONLY clause with the TRANSACTION LOG RENAME or TRANSACTION LOG TRUNCATE clauses.

TRANSACTION LOG ONLY clause You can specify the TRANSACTION LOG ONLY clause to create a backup copy of the transaction log, without copying the other database files.

TRANSACTION LOG RENAME [ MATCH ] clause This clause causes the database server to rename the current transaction log at the completion of the backup. If the MATCH keyword is omitted, the backup copy of the transaction log has the same name as the current transaction log for the database. If you supply the MATCH keyword, the backup copy of the transaction log is given a name of the form YYMMDnm.log, to match the renamed copy of the current transaction log. Using the MATCH keyword enables the same statement to be executed several times without writing over old data.

The transaction log can be renamed without completing a backup by specifying an empty directory name with the TRANSACTION LOG ONLY clause. For example:
**TRANSACTION LOG TRUNCATE clause**  
If this clause is used, the current transaction log is truncated and restarted at the completion of the backup. Do not use this clause if you are using database mirroring.

The transaction log can be truncated without completing a backup by specifying an empty directory name with the TRANSACTION LOG ONLY clause. For example:

```
BACKUP DATABASE DIRECTORY ''
TRANSACTION LOG ONLY
TRANSACTION LOG TRUNCATE;
```

**archive-root clause**  
The file name or tape drive device name for the archive file.

To back up to tape, you must specify the device name of the tape drive. The number automatically appended to the end of the archive file name is incremented each time you execute an archive backup.

The backslash (\) is an escape character in SQL strings, so each backslash must be doubled.

**ON EXISTING ERROR clause**  
This clause applies only to image backups. By default, existing files are overwritten when you execute a BACKUP DATABASE statement. If this clause is used, an error occurs if any of the files to be created by the backup already exist.

**ATTENDED clause**  
The clause applies only when backing up to a tape device. ATTENDED ON (the default) indicates that someone is available to monitor the status of the tape drive and to place a new tape in the drive when needed. A message is sent to the application that issued the BACKUP DATABASE statement if the tape drive requires intervention. The database server then waits for the drive to become ready. This may happen, for example, when a new tape is required.

If ATTENDED OFF is specified and a new tape is required or the drive is not ready, no message is sent and an error is given.

**WITH COMMENT clause**  
This clause records a comment in the backup history file. For archive backups, the comment is also recorded in the archive file.

**HISTORY clause**  
This clause enables or disables backup history. By default, this clause is ON, meaning that each backup operation appends a line to the backup.syb file. Specifying HISTORY OFF prevents updates to the backup.syb file, and is recommended when:

- The database is backed up frequently.
- There is no procedure in place to periodically archive or delete the backup.syb file.
- Disk space is limited.

**AUTO TUNE WRITERS clause**  
Specifying this clause enables or disables the automatic tuning of writers. During the backup process, one writer writes the backup files to the backup directory. If the backup directory is on a device that can handle an increased writer load (such as a RAID array), the default AUTO TUNE WRITERS ON improves overall backup performance by increasing the number of writers. The database server periodically examines the read and write performances of all devices that are...
participating in the backup. Specifying AUTO TUNE WRITERS OFF prevents the database server from creating additional writers.

**WITH CHECKPOINT LOG clause** This clause specifies how the backup processes the database files before writing them to the destination directory. The choice of whether to apply pre-images during a backup, or copy the checkpoint log as part of the backup, has performance implications. The default setting is AUTO for image backups and COPY for archive backups.

- **COPY clause** This option cannot be used with the WAIT BEFORE START clause of the BACKUP statement.

  When you specify COPY, the backup reads the database files without applying any modified pages. The entire checkpoint log and the system dbspace are copied to the backup directory. The next time the database server is started, the database server automatically recovers the database to the state it was in as of the checkpoint at the time the backup started.

  Because pages do not have to be written to the temporary file, using this option can provide better backup performance, and reduce internal server contention for other connections that are operating during a backup. However, since the checkpoint log contains original images of modified pages, it grows in the presence of database updates. With copy specified, the backed-up copy of the database files may be larger than the database files at the time the backup started. The COPY option should be used if disk space in the destination directory is not an issue.

- **NO COPY clause** When you specify NO COPY, the checkpoint log is not copied as part of the backup. This option causes modified pages to be saved in the temporary file so that they can be applied to the backup as it progresses. The backup copies of the database files will be the same size as the database when the backup operation commenced.

  This option results in smaller backed up database files, but the backup may proceed more slowly, and possibly decrease performance of other operations in the database server. It is useful in situations where space on the destination drive is limited.

- **RECOVER clause** When you specify RECOVER, the database server copies the checkpoint log (as with the COPY option), but applies the checkpoint log to the database when the backup is complete. This restores the backed up database files to the same state (and size) that they were in at the start of the backup operation. This option is useful if space on the backup drive is limited (it requires the same amount of space as the COPY option for backing up the checkpoint log, but the resulting file size is smaller).

- **AUTO clause** When you specify AUTO, the database server checks the amount of available disk space on the volume hosting the backup directory. If there is at least twice as much disk space available as the size of the database at the start of the backup, then this option behaves as if copy were specified. Otherwise, it behaves as NO COPY. AUTO is the default behavior.

**MAX WRITE clause** For archive backups, by default one thread is dedicated to writing the backup files. If the backup directory is on a device that can handle an increased writer load (such as a RAID array), then overall backup performance can be improved by increasing the number of threads acting as writers.
If AUTO is specified, one output stream is created for each reader thread. The value \( n \) specifies the maximum number of output streams that can be created, up to the number of reader threads. The default value for this clause is 1. If you are backing up to tape, only one writer can be used.

The first stream, stream 0, produces files named \( myarchive.X \), where \( X \) is a number that starts at 1 and continues incrementing to the number of files required. All of the other streams produce files named \( myarchive.Y.Z \), where \( Y \) is the stream number (starting at 1), and \( Z \) is a number that starts at 1 and continues incrementing to the number of files required.

**FREE PAGE ELIMINATION clause**

By default, archive backups skip some free pages, which can result in smaller and potentially faster backups. Free page elimination has no effect on the back up of transaction log files because transaction log files do not contain free pages. Databases with large transaction log files may not benefit as much from free page elimination as databases with small transaction log files.

When you back up a strongly-encrypted database with free page elimination turned on, you must specify the encryption key when restoring the database. When you back up a strongly-encrypted database with free page elimination turned off, you do not need to specify the encryption key when restoring the database.

As of version 12, you cannot restore archive backups created with version 11 or earlier database servers.

**Remarks**

The BACKUP statement performs a server-side backup. To perform a client-side backup, use the dbbackup utility.

If the disk sandbox feature is enabled for the database, you must specify a secure feature key that disables the disk sandbox feature for the database server to be able to make the backup in a directory outside of the sandbox (the directory where the main database file is located and any subdirectories of this directory). See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

Each backup operation, whether image or archive, updates a history file called \( backup.syb \). This file records the BACKUP and RESTORE operations that have been performed on a database server. For information about how the location of the \( backup.syb \) file is determined, see “SALOGDIR environment variable” [SQL Anywhere Server - Database Administration].

To create a backup that can be started on a read-only server without having to go through recovery, you must use both the WAIT BEFORE START and WITH CHECKPOINT LOG NO COPY clauses. The WAIT BEFORE START clause ensures that the rollback log is empty, and the WITH CHECKPOINT LOG NO COPY clause ensures that the checkpoint log is empty. If either of these files is missing, then recovery is required. You can use WITH CHECKPOINT LOG RECOVER as an alternative to the WAIT BEFORE START and WITH CHECKPOINT LOG NO COPY clauses if you do not need to recover the database you backed up.

- **Syntax 1 - Image backup**

  An image backup creates copies of each of the database files, in the same way as the Backup utility (dbbackup). By default, the Backup utility makes the backup on the client computer, but you can specify the -s option to create the backup on the database server when using the Backup utility. For the BACKUP DATABASE statement, however, the backup can only be made on the database server.
Optionally, only the database file(s) or transaction log can be saved. The transaction log may also be renamed or truncated after the backup has completed.

Alternatively, you can specify an empty string as a directory to rename or truncate the log without copying it first. This is useful in a replication environment where space is a concern. You can use this feature with an event handler on transaction log size to rename the transaction log when it reaches a given size, and with the delete_old_logs option to delete the transaction log when it is no longer needed.

To restore from an image backup, copy the saved files back to their original locations and reapply the transaction logs.

- **Syntax 2 - Archive backup**  An archive backup creates a single file holding all the required backup information. The destination can be either a file name or a tape drive device name.

There can be only one backup on a given tape. The tape is ejected at the end of the backup.

Only one archive per tape is allowed, but a single archive can span multiple tapes. To restore a database from an archive backup, use the RESTORE DATABASE statement.

If a RESTORE DATABASE statement references an archive file containing only a transaction log, the statement must specify a file name for the location of the restored database file, even if that file does not exist. For example, to restore from an archive that only contains a transaction log to the directory `C:\MYNEWDB`, the RESTORE DATABASE statement is:

```
RESTORE DATABASE 'c:\temp\mynewdb\my.db' FROM archive-root
```

Caution
Backup copies of the database and transaction log must not be changed in any way. If there were no transactions in progress during the backup, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG RECOVER or WITH CHECKPOINT LOG NO COPY, you can check the validity of the backup database using read-only mode or by validating a copy of the backup database.

However, if transactions were in progress, or if you specified BACKUP DATABASE WITH CHECKPOINT LOG COPY, the database server must perform recovery on the database when you start it. Recovery modifies the backup copy, which is not desirable.

During the execution of this statement, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

**Privileges**
You must have the BACKUP DATABASE system privilege.

**Side effects**
Causes a checkpoint.
See also

- “Creating a server-side backup” [SQL Anywhere Server - Database Administration]
- “Database recovery with multiple transaction logs” [SQL Anywhere Server - Database Administration]
- “Progress connection property” [SQL Anywhere Server - Database Administration]
- “progress_messages option” [SQL Anywhere Server - Database Administration]
- “Backup utility (dbbackup)” [SQL Anywhere Server - Database Administration]
- “Image backups” [SQL Anywhere Server - Database Administration]
- “Strings” on page 6
- “RESTORE DATABASE statement” on page 937
- “KEY CLAUSE, RESTORE DATABASE statement” on page 938
- “Backup and data recovery” [SQL Anywhere Server - Database Administration]
- “Transaction log file management in a database mirroring system” [SQL Anywhere Server - Database Administration]
- “EXECUTE IMMEDIATE statement [SP]” on page 791
- “Parallel database backups” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.
- Windows Mobile Only the BACKUP DATABASE DIRECTORY syntax (syntax 1) is supported on Windows Mobile.

Example

Back up the current database and the transaction log, each to a different file, and rename the existing transaction log. An image backup is created.

```
BACKUP DATABASE
DIRECTORY 'c:\temp\backup'
TRANSACTION LOG RENAME;
```

The option to rename the transaction log is useful, especially in replication environments where the old transaction log is still required.

Back up the current database and transaction log to tape:

```
BACKUP DATABASE
TO '\\\.\tape0';
```

Rename the transaction log without making a copy:

```
BACKUP DATABASE DIRECTORY ''
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

Execute the BACKUP DATABASE statement with a dynamically-constructed directory name:

```
CREATE EVENT NightlyBackup
SCHEDULE
START TIME '23:00' EVERY 24 HOURS
HANDLER
BEGIN
```
DECLARE dest LONG VARCHAR;
DECLARE day_name CHAR(20);

SET day_name = DATENAME( WEEKDAY, CURRENT DATE );
SET dest = 'd:\backups\'|| day_name;
    BACKUP DATABASE DIRECTORY dest
    TRANSACTION LOG RENAME;
END;

BEGIN SNAPSHOT statement

Starts a snapshot at a specified period in time for use with snapshot isolation transactions.

Syntax

BEGIN SNAPSHOT

Remarks

By default, when a transaction begins, the database server defers creating the snapshot until the
application causes the first row of a table to be fetched. You can use the BEGIN SNAPSHOT statement to
start the snapshot earlier within the transaction. The database server creates a snapshot when the BEGIN
SNAPSHOT statement is executed by a snapshot transaction.

The statement fails and returns an error when either of the following conditions is met:

● support for snapshots transactions has not been enabled for the database.

● a snapshot has already been started for the current transaction.

This statement is also useful for non-snapshot transactions because it allows them to start a snapshot that
can be used later in the transaction for a statement-level snapshot operation.

Privileges

None.

Side effects

None.

See also

● “allow_snapshot_isolation option” [SQL Anywhere Server - Database Administration]
● “Snapshot isolation” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

● SQL/2008 Vendor extension.

BEGIN statement

Specify a compound statement.
Syntax

```sql
[ statement-label : ]
BEGIN [[ NOT ] ATOMIC ]
[ local-declaration; ... ]
statement-list
[ EXCEPTION [ exception-case ... ] ]
END [ statement-label ]
```

- `local-declaration`: variable-declaration
cursor-declaration
exception-declaration
temporary-table-declaration

- `exception-case`:

```sql
WHEN exception-name [, ... ] THEN statement-list
| WHEN OTHERS THEN statement-list
```

- `variable-declaration` and `exception-declaration`: See “DECLARE statement” on page 736.
- `cursor-declaration`: See “DECLARE CURSOR statement [ESQL] [SP]” on page 729.
- `temporary-table-declaration`: See “DECLARE LOCAL TEMPORARY TABLE statement” on page 735.

Parameters

- **statement-label**: If the ending `statement-label` is specified, it must match the beginning `statement-label`. The LEAVE statement can be used to resume execution at the first statement after the compound statement. The compound statement that is the body of a procedure or trigger has an implicit label that is the same as the name of the procedure or trigger.

- **ATOMIC clause**: An atomic statement is a statement that is executed completely or not at all. For example, an UPDATE statement that updates thousands of rows might encounter an error after updating many rows. If the statement does not complete, all changes revert back to their original state. Similarly, if you specify that the BEGIN statement is atomic, the statement is executed either in its entirety or not at all.

- **local-declaration**: Immediately following the BEGIN, a compound statement can have local declarations for objects that only exist within the compound statement. A compound statement can have a local declaration for a variable, a cursor, a temporary table, or an exception. Local declarations can be referenced by any statement in that compound statement, or in any compound statement nested within it. Local declarations of the compound statement are visible to the exception handlers for the statement. Local declarations are not visible to other procedures that are called from within a compound statement.

Remarks

The body of a procedure or trigger is a compound statement. Compound statements can also be used in control statements within a procedure or trigger.

A compound statement allows one or more SQL statements to be grouped together and treated as a unit. A compound statement starts with the keyword BEGIN and ends with the keyword END.
Privileges

None.

Side effects

None.

See also

- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]
- “Error and warning handling” [SQL Anywhere Server - SQL Usage]
- “Exception handling and atomic compound statements” [SQL Anywhere Server - SQL Usage]
- “Exception handlers” [SQL Anywhere Server - SQL Usage]
- “Atomic compound statements” [SQL Anywhere Server - SQL Usage]
- “CONTINUE statement” on page 546
- “SIGNAL statement [SP]” on page 992
- “RESIGNAL statement [SP]” on page 936
- “RAISERROR statement” on page 920
- “BEGIN statement [TSQL]” on page 526

Standards and compatibility

- SQL/2008  BEGIN, which identifies a compound statement, comprises part of optional SQL language feature P002 in SQL/2008.

Example

The body of a procedure or trigger is a compound statement.

```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
  DECLARE err_notfound EXCEPTION FOR
  SQLSTATE '02000';
  DECLARE curThisCust CURSOR FOR
  SELECT CompanyName, CAST(
    sum( SalesOrderItems.Quantity *
    Products.UnitPrice ) AS INTEGER) VALUE
  FROM GROUPO.Customers
  LEFT OUTER JOIN SalesOrders
  LEFT OUTER JOIN SalesOrderItems
  LEFT OUTER JOIN Products
  GROUP BY CompanyName;
  DECLARE ThisValue INT;
  DECLARE ThisCompany CHAR( 35 );
  SET TopValue = 0;
  OPEN curThisCust;
  CustomerLoop:
  LOOP
    FETCH NEXT curThisCust
    INTO ThisCompany, ThisValue;
    IF SQLSTATE = err_notfound THEN
      LEAVE CustomerLoop;
    END IF;
    IF ThisValue > TopValue THEN
      SET TopValue = ThisValue;
      SET TopCompany = ThisCompany;
    END IF;
```
END LOOP CustomerLoop;
CLOSE curThisCust;
END;

The example below declares the following variables:

- v1 as an INT with the initial setting of 5.
- v2 and v3 as CHAR(10), both with an initial value of abc.

```
BEGIN
    DECLARE v1 INT = 5
    DECLARE v2, v3 CHAR(10) = 'abc'
    // ...
END
```

**BEGIN statement [TSQL]**

Specifies a compound statement.

**Syntax**

```
BEGIN
statement-list
END
```

- **statement-list**: A list of statements and declarations.
- **sql-statement**
  - **variable-declaration**: See “DECLARE statement” on page 736.
  - **cursor-declaration**: See “DECLARE CURSOR statement [ESQL] [SP]” on page 729.
  - **temporary-table-declaration**: See “DECLARE LOCAL TEMPORARY TABLE statement” on page 735.

**Parameters**

- **statement-list**: A list of statements and declarations.

**Remarks**

A BEGIN statement allows one or more SQL statements to be grouped together and treated as a unit, and starts with the keyword BEGIN and ends with the keyword END.

Error handling is different for Transact-SQL compound statements.

**Privileges**

None.

**Side effects**

None.
Standards and compatibility

- **SQL/2008**  
  BEGIN, which identifies a compound statement, comprises part of optional SQL language feature P002 in SQL/2008.

Example

The example below declares the following variables:

- v1 as an INT with the initial setting of 5.
- v2 and v3 as CHAR(10), both with an initial value of abc.

```sql
BEGIN
  DECLARE v1 INT = 5
  DECLARE v2, v3 CHAR(10) = 'abc'
  // ...
END
```

**BEGIN TRANSACTION statement [T-SQL]**

Begins a user-defined transaction.

Syntax

```sql
BEGIN TRAN[SACTION] [transaction-name]
```

Remarks

The optional parameter `transaction-name` is the name assigned to this transaction. It must be a valid identifier. Use transaction names only on the outermost pair of nested BEGIN/COMMIT or BEGIN/ROLLBACK statements.

When executed inside a transaction, the `BEGIN TRANSACTION` statement increases the nesting level of transactions by one. The nesting level is decreased by a COMMIT statement. When transactions are nested, only the outermost COMMIT makes the changes to the database permanent.

Both Adaptive Server Enterprise and SQL Anywhere have two transaction modes.

The default Adaptive Server Enterprise transaction mode, called unchained mode, commits each statement individually, unless an explicit `BEGIN TRANSACTION` statement is executed to start a transaction. In contrast, the ISO SQL/2008 compatible chained mode only commits a transaction when an explicit COMMIT is executed or when a statement that carries out an autocommit (such as a data definition statement) is executed.
You can control the mode by setting the chained database option. The default setting for ODBC and embedded SQL connections in SQL Anywhere is On, in which case SQL Anywhere runs in chained mode. (ODBC users should also check the AutoCommit ODBC setting). The default for TDS connections is Off.

In unchained mode, a transaction is implicitly started before any data retrieval or manipulation statement. These statements include: DELETE, INSERT, OPEN, FETCH, SELECT, and UPDATE. You must still explicitly end the transaction with a COMMIT or ROLLBACK statement.

You cannot alter the setting of the chained option within a transaction.

**Caution**
When calling a stored procedure, you should ensure that it operates correctly under the required transaction mode.

The current nesting level is held in the global variable @@trancount. The @@trancount variable has a value of zero before the first BEGIN TRANSACTION statement is executed, and only a COMMIT executed when @@trancount is equal to one makes changes to the database permanent.

You should not rely on the value of @@trancount for more than keeping track of the number of explicit BEGIN TRANSACTION statements that have been executed.

When Adaptive Server Enterprise starts a transaction implicitly, the @@trancount variable is set to 1. SQL Anywhere does not set the @@trancount value to 1 when a transaction is started implicitly. Instead, the SQL Anywhere @@trancount variable has a value of zero before any BEGIN TRANSACTION statement (even though there is a current transaction), while in Adaptive Server Enterprise (in chained mode) it has a value of 1.

For transactions starting with a BEGIN TRANSACTION statement, @@trancount has a value of 1 in both SQL Anywhere and Adaptive Server Enterprise after the first BEGIN TRANSACTION statement. If a transaction is implicitly started with a different statement, and a BEGIN TRANSACTION statement is then executed, @@trancount has a value of 2 in both SQL Anywhere, and Adaptive Server Enterprise after the BEGIN TRANSACTION statement.

A ROLLBACK statement without a transaction or savepoint name always rolls back statements to the outermost BEGIN TRANSACTION (explicit or implicit) statement, and cancels the entire transaction.

**Privileges**
None.

**Side effects**
None.
See also

- “COMMIT statement” on page 541
- “ROLLBACK statement” on page 950
- “SAVEPOINT statement” on page 954
- “isolation_level option” [SQL Anywhere Server - Database Administration]
- “chained option” [SQL Anywhere Server - Database Administration]
- “Savepoints within transactions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 Vendor extension.
- Transact-SQL BEGIN TRANSACTION is supported by Adaptive Server Enterprise.

Example

The following batch reports successive values of @@trancount as 0, 1, 2, 1, and 0. The values are printed in the database server messages window.

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT
PRINT @@trancount
COMMIT
PRINT @@trancount
```

**BREAK statement [T-SQL]**

Exits a compound statement or loop.

**Syntax**

```
BREAK
```

**Remarks**

The BREAK statement is a control statement that allows you to leave a loop. Execution resumes at the first statement after the loop.

**Privileges**

None.

**Side effects**

None.
See also

- “WHILE statement [T-SQL]” on page 1050
- “CONTINUE statement” on page 546
- “BEGIN statement” on page 523
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 Transact-SQL extension.

Example

In this example, the BREAK statement breaks the WHILE loop if the most expensive product has a price above $50. Otherwise, the loop continues until the average price is greater than or equal to $30:

```sql
WHILE ( SELECT AVG( UnitPrice ) FROM Products ) < $30
BEGIN
    UPDATE GROUPO.Products
    SET UnitPrice = UnitPrice + 2
    IF ( SELECT MAX(UnitPrice) FROM Products ) > $50
        BREAK
END
```

**CALL statement**

Invokes a procedure.

**Syntax 1**

```sql
[variable = ] CALL procedure-name ([ expression, ... ])
```

**Syntax 2**

```sql
[variable = ] CALL procedure-name ([ parameter-name = expression, ... ])
```

**Remarks**

The CALL statement invokes a procedure that has been previously created with a CREATE PROCEDURE statement. When the procedure completes, any INOUT or OUT parameter value is copied back.

The argument list can be specified by position or by using keyword format. By position, the arguments match up with the corresponding parameter in the parameter list for the procedure (DEFAULT can be used for an optional parameter). By keyword, the arguments are matched up with the named parameters.

Procedure arguments can be assigned default values in the CREATE PROCEDURE statement, and missing parameters are assigned the default value. If no default is set, and an argument is not provided, an error is given.

Inside a procedure, a CALL statement can be used in a DECLARE statement when the procedure returns result sets.
Subqueries and spatial method calls are not allowed as arguments to a stored procedure in a CALL statement.

Procedures can return an integer value (for example, as a status indicator) using the RETURN statement. You can save this return value in a variable using the equality sign as an assignment operator:

If the procedure being called returns an INT and the value is NULL, then the error status value, 0, is returned instead. There is no way to differentiate between this case and the case of an actual value of 0 being returned.

Note
Use of this statement to invoke a function is deprecated. If you have a function you want to call, consider using an assignment statement to invoke the function and assign its result to a variable. For example:

```
DECLARE varname INT;
SET varname=test( );
```

Privileges
You must be the owner of the procedure, or have one of the following privileges:

- EXECUTE privilege on the procedure
- EXECUTE ANY PROCEDURE system privilege

Side effects
None.

See also
- “CREATE FUNCTION statement” on page 594
- “CREATE FUNCTION statement [External call]” on page 580
- “CREATE FUNCTION statement [Web service]” on page 586
- “CREATE PROCEDURE statement” on page 639
- “CREATE PROCEDURE statement [External call]” on page 620
- “CREATE PROCEDURE statement [Web service]” on page 628
- “EXECUTE statement [T-SQL]” on page 796
- “Result sets” [SQL Anywhere Server - SQL Usage]
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008 Core feature. The use of the RETURN statement to return a value from a stored procedure is a vendor extension; SQL/2008 supports return values only for SQL-invoked functions, not for procedures. Default values for stored procedure arguments are not supported in SQL/2008.

Example
This example creates a procedure to return the number of orders placed by the customer whose ID is supplied, creates a variable to hold the result, calls the procedure, and displays the result.
CASE statement

Selects an execution path based on multiple cases.

Syntax 1

```
CASE value-expression
    WHEN [ constant | NULL ] THEN statement-list ...
    WHEN [ constant | NULL ] THEN statement-list ] ...
    [ ELSE statement-list ]
END [ CASE ]
```

Syntax 2

```
CASE
    WHEN [ search-condition | NULL] THEN statement-list ...
    WHEN [ search-condition | NULL] THEN statement-list ] ...
    [ ELSE statement-list ]
END [ CASE ]
```

Remarks

**Syntax 1**  The CASE statement is a control statement that allows you to choose a list of SQL statements to execute based on the value of an expression. The `value-expression` is an expression that takes on a single value, which may be a string, a number, a date, or other SQL data type. If a WHEN clause exists for the value of `value-expression`, the `statement-list` in the WHEN clause is executed. If no appropriate WHEN clause exists, and an ELSE clause exists, the `statement-list` in the ELSE clause is executed. Execution resumes at the first statement after the END CASE.

If the `value-expression` can be null, use the ISNULL function to replace the NULL `value-expression` with a different expression.

**Syntax 2**  With this form, the statements are executed for the first satisfied `search-condition` in the CASE statement. The ELSE clause is executed if none of the `search-conditions` are met.

If the expression can be NULL, use the following syntax for the first `search-condition`:

```
WHEN search-condition IS NULL THEN statement-list
```
Note
Do not confuse the syntax of the CASE statement with that of the CASE expression.

Privileges
None.

Side effects
None.

See also
- “ISNULL function [Miscellaneous]” on page 277
- “Unknown values: NULL” [SQL Anywhere Server - SQL Usage]
- “BEGIN statement” on page 523
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]
- “CASE expressions” on page 24
- “CASE statement [T-SQL]” on page 534

Standards and compatibility
- SQL/2008 The CASE statement is part of language feature P002 (Computational completeness) of the SQL/2008 standard. The use of END alone, rather than END CASE, is a vendor extension.
- Transact-SQL The CASE statement is supported by Adaptive Server Enterprise.

Example
The following procedure using a case statement classifies the products listed in the Products table of the SQL Anywhere sample database into one of shirt, hat, shorts, or unknown.

```sql
CREATE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
    DECLARE prod_name CHAR(20);
    SELECT Name INTO prod_name FROM GROUPO.Products
    WHERE ID = product_ID;
    CASE prod_name
    WHEN 'Tee Shirt' THEN
        SET type = 'Shirt'
    WHEN 'Sweatshirt' THEN
        SET type = 'Shirt'
    WHEN 'Baseball Cap' THEN
        SET type = 'Hat'
    WHEN 'Visor' THEN
        SET type = 'Hat'
    WHEN 'Shorts' THEN
        SET type = 'Shorts'
    ELSE
        SET type = 'UNKNOWN'
    END CASE;
END;
```

The following example uses Syntax 2 to generate a message about product quantity within the SQL Anywhere sample database.
CREATE PROCEDURE StockLevel (IN product_ID INT)
BEGIN
  DECLARE qty INT;
  SELECT Quantity INTO qty FROM GROUPO.Products
  WHERE ID = product_ID;
  CASE
    WHEN qty < 30 THEN
      MESSAGE 'Order Stock' TO CLIENT;
    WHEN qty > 100 THEN
      MESSAGE 'Overstocked' TO CLIENT;
    ELSE
      MESSAGE 'Sufficient stock on hand' TO CLIENT;
  END CASE;
END;

CASE statement [T-SQL]

Selects an execution path based on multiple cases.

Syntax 1

```
CASE value-expression
  WHEN [ constant | NULL ] THEN statement-list ... 
  [ WHEN [ constant | NULL ] THEN statement-list ] ... 
  [ ELSE statement-list ]
END
```

Syntax 2

```
CASE
  WHEN [ search-condition | NULL ] THEN statement-list ... 
  [ WHEN [ search-condition | NULL ] THEN statement-list ] ... 
  [ ELSE statement-list ]
END
```

Remarks

Syntax 1 The CASE statement is a control statement that allows you to choose a list of SQL statements to execute based on the value of an expression. The value-expression is an expression that takes on a single value, which may be a string, a number, a date, or other SQL data type. If a WHEN clause exists for the value of value-expression, the statement-list in the WHEN clause is executed. If no appropriate WHEN clause exists, and an ELSE clause exists, the statement-list in the ELSE clause is executed. Execution resumes at the first statement after the END CASE.

If the value-expression can be null, use the ISNULL function to replace the NULL value-expression with a different expression.

Syntax 2 With this form, the statements are executed for the first satisfied search-condition in the CASE statement. The ELSE clause is executed if none of the search-conditions are met.

If the expression can be NULL, use the following syntax for the first search-condition:

```
WHEN search-condition IS NULL THEN statement-list
```
Note
Do not confuse the syntax of the CASE statement with that of the CASE expression.

Privileges
None.

Side effects
None.

See also
- “ISNULL function [Miscellaneous]” on page 277
- “BEGIN statement” on page 523
- “CASE expressions” on page 24
- “Unknown values: NULL” [SQL Anywhere Server - SQL Usage]
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- **SQL/2008** The CASE statement is part of language feature P002 (Computational completeness) of the SQL/2008 standard. The SQL standard requires END CASE to terminate the CASE statement, rather than END alone.

- **Transact-SQL** Compatible with Adaptive Server Enterprise.

Example
The following procedure using a case statement classifies the products listed in the Products table of the SQL Anywhere sample database into one of shirt, hat, shorts, or unknown.

```
CREATE PROCEDURE DBA.ProductType( @product_ID INTEGER, @TYPE CHAR(10) OUTPUT ) AS
BEGIN
    DECLARE @prod_name CHAR(20)
    SELECT Name INTO @prod_name FROM GROUPO.Products
    WHERE ID = @product_ID
    IF @prod_name = 'Tee Shirt'
        SET @TYPE = 'Shirt'
    ELSE IF @prod_name = 'Sweatshirt'
        SET @TYPE = 'Shirt'
    ELSE IF @prod_name = 'Baseball Cap'
        SET @TYPE = 'Hat'
    ELSE IF @prod_name = 'Visor'
        SET @TYPE = 'Hat'
    ELSE IF @prod_name = 'Shorts'
        SET @TYPE = 'Shorts'
    ELSE
        SET @TYPE = 'UNKNOWN'
END;
```
The following example uses Syntax 2 to generate a message about product quantity within the SQL Anywhere sample database.

```
CREATE PROCEDURE DBA.StockLevel( @product_ID INTEGER ) AS
BEGIN
    DECLARE @qty INTEGER
    SELECT Quantity INTO @qty FROM GROUPO.Products
    WHERE ID = @product_ID
    IF @qty < 30
        MESSAGE 'Order Stock' TO CLIENT
    ELSE IF @qty > 100
        MESSAGE 'Overstocked' TO CLIENT
    ELSE
        MESSAGE 'Sufficient stock on hand' TO CLIENT
END;
```

CHECKPOINT statement

Checkpoints the database.

Syntax

```
CHECKPOINT
```

Remarks

The CHECKPOINT statement forces the database server to execute a checkpoint. Checkpoints are also performed automatically by the database server according to an internal algorithm. It is not normally required for applications to issue the CHECKPOINT statement.

Privileges

You must have the CHECKPOINT system privilege to perform a checkpoint on a database running on a network server (dbsrv).

No privileges are required to perform a checkpoint on a database running on a personal database server (dbeng16).

Side effects

None.

See also

- “Backup and data recovery” [SQL Anywhere Server - Database Administration]
- “Checkpoint logs” [SQL Anywhere Server - Database Administration]
- “checkpoint_time option” [SQL Anywhere Server - Database Administration]
- “recovery_time option” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- **SQL/2008** Vendor extension.
- **Transact-SQL** The CHECKPOINT statement is supported by Adaptive Server Enterprise.
CLEAR statement [Interactive SQL]
Closes any open result sets in Interactive SQL.

Syntax
CLEAR

Remarks
Closes any open result sets and leaves the contents of the SQL Statements pane unchanged.

Privileges
None.

Side effects
Closes the cursor associated with the data being cleared.

See also
● “Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility
● SQL/2008   Vendor extension.

CLOSE statement [ESQL] [SP]
Closes a cursor.

Syntax
CLOSE cursor-name

cursor-name : identifier | hostvar

Remarks
This statement closes the named cursor.

The cursor must have been previously opened.

Privileges
None.

Side effects
None.
Standards and compatibility

- **SQL/2008** Core feature. When used in embedded SQL, the CLOSE statement is part of optional language feature B031 (Basic dynamic SQL).

- **Transact-SQL** Supported by Adaptive Server Enterprise.

Example

The following examples close cursors in embedded SQL.

```sql
EXEC SQL CLOSE employee_cursor;
EXEC SQL CLOSE :cursor_var;
```

The following procedure uses a cursor.

```sql
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
    DECLARE err_notfound EXCEPTION
        FOR SQLSTATE '02000';
    DECLARE curThisCust CURSOR FOR
        SELECT CompanyName, CAST(    sum(SalesOrderItems.Quantity * Products.UnitPrice) AS INTEGER) VALUE
        FROM GROUPO.Customers
        LEFT OUTER JOIN SalesOrders
        LEFT OUTER JOIN SalesOrderItems
        LEFT OUTER JOIN Products
        GROUP BY CompanyName;
    DECLARE ThisValue INT;
    DECLARE ThisCompany CHAR(35);
    SET TopValue = 0;
    OPEN curThisCust;
    CustomerLoop:
    LOOP
        FETCH NEXT curThisCust
        INTO ThisCompany, ThisValue;
        IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop;
        END IF;
        IF ThisValue > TopValue THEN
            SET TopValue = ThisValue;
            SET TopCompany = ThisCompany;
        END IF;
    END LOOP CustomerLoop;
    CLOSE curThisCust;
END
```

**COMMENT statement**

Stores a comment for a database object in the system tables.
Syntax

```
COMMENT ON {
  COLUMN [ owner.][table-name.column-name
  CERTIFICATE certificate-name
  DBSPACE dbspace-name
  EVENT [ owner.][event-name
  EXTERNAL ENVIRONMENT environment-name
  EXTERNAL [ ENVIRONMENT ] OBJECT object-name
  FOREIGN KEY [ owner.][table-name.key-name
  INDEX [ [ owner.][table.][index-name
  INTEGRATED LOGIN integrated-login-id
  JAVA CLASS java-class-name
  JAVA JAR java-jar-name
  KERBEROS LOGIN "client-Kerberos-principal"
  LDAP SERVER ldapua-server-name
  LOGIN POLICY policy-name
  MATERIALIZED VIEW [ owner.][materialized-view-name
  MIRROR SERVER mirror-server-name
  PRIMARY KEY ON [ owner.][table-name
  PROCEDURE [ owner.][procedure-name
  PUBLICATION [ owner.][publication-name
  REMOTE MESSAGE TYPE remote-message-type-name
  ROLE role-name
  SEQUENCE sequence-name
  SERVICE web-service-name
  SPATIAL REFERENCE SYSTEM srs-name
  SPATIAL UNIT OF MEASURE uom-identifier
  SYNCHRONIZATION PROFILE synchronization-profile-name
  TABLE [ owner.][table-name
  TEXT CONFIGURATION [ owner.][text-config-name
  TEXT INDEX text-index-name ON [ owner.][table-name
  TRIGGER [ [ owner.][tablename.][trigger-name
  USER userid
  VIEW [ owner.][view-name
  }

  IS comment

  comment : string | NULL
```

environment-name :

JAVA
  PERL
  PHP
  CLR
  C_ESQL32
  C_ESQL64
  C_ODBC32
  C_ODBC64

Remarks

The COMMENT statement allows you to set a remark (comment) for an object in the database. The COMMENT statement updates remarks listed in the ISYSREMARK system table. You can remove a comment by setting it to NULL. For a comment on an index or trigger, the owner of the comment is the owner of the table on which the index or trigger is defined.
You cannot add comments for local temporary tables.

If you use the **Database Documentation Wizard** to document your SQL Anywhere database, you have the option to include the comments for procedures, functions, triggers, events, and views in the output.

**Privileges**

If you have the COMMENT ANY OBJECT system privilege, you can comment on any you can create with the CREATE ANY OBJECT system privilege. If you do not have the COMMENT ANY OBJECT system privilege, you must have the equivalent as noted below:

- For database objects, at least one of the following must be true:
  - you own the object
  - you have the ability to create or alter objects of the same type owned by other users (for example, CREATE ANY TABLE, or ALTER ANY OBJECT)
  - you have the ability to manage objects of that type (for example, MANAGE ANY USER)

- For system roles, you must have administrative privilege over the role.

- For user-defined roles, you must have the MANAGE ROLES system privilege, or have administrative privilege over the role.

- For Kerberos or integrated logins, you must have the MANAGE ANY USER system privilege.

- For Java classes or jars, you must have the MANAGE ANY EXTERNAL OBJECT system privilege.

**Side effects**

Automatic commit.

**Standards and compatibility**

- **SQL/2008**  Vendor extension.

- **Transact-SQL**  Not supported by Adaptive Server Enterprise.

**Example**

The following examples show how to add and remove a comment.

1. Add a comment to the Employees table.

   ```sql
   COMMENT ON TABLE GROUPO.Employees
   IS 'Employee information';
   ```

2. Remove the comment from the Employees table.

   ```sql
   COMMENT
   ON TABLE GROUPO.Employees
   IS NULL;
   ```
To view the comment set for an object, use a SELECT statement similar to the following. This example retrieves the comment set for the ViewSalesOrders view in the SQL Anywhere sample database.

```sql
SELECT remarks
FROM SYSTAB t, SYSREMARK r
WHERE t.object_id = r.object_id
AND t.table_name = 'ViewSalesOrders';
```

See also

- “Documenting a database” [SQL Anywhere Server - Database Administration]

**COMMIT statement**

Makes changes to the database permanent, or terminates a user-defined transaction.

**Syntax 1**

```sql
COMMIT [ WORK ]
```

**Syntax 2**

```sql
COMMIT TRAN[SACTION] [ transaction-name ]
```

**Parameters**

- `transaction-name`  An optional name assigned to this transaction. It must be a valid identifier. You should use transaction names only on the outermost pair of nested BEGIN/COMMIT or BEGIN/ROLLBACK statements.

For more information about transaction nesting in Adaptive Server Enterprise and SQL Anywhere, see “BEGIN TRANSACTION statement [T-SQL]” on page 527.

You can use a set of options to control the detailed behavior of the COMMIT statement. See:

- “cooperative_commit_timeout option” [SQL Anywhere Server - Database Administration]
- “cooperative_commits option” [SQL Anywhere Server - Database Administration]
- “delayed_commits option” [SQL Anywhere Server - Database Administration]
- “delayed_commit_timeout option” [SQL Anywhere Server - Database Administration]
- “Permanent data changes” [SQL Anywhere Server - SQL Usage]

You can use the Commit connection property to return the number of commits on the current connection.

**Remarks**

- **Syntax 1**  The COMMIT statement ends a transaction and makes all changes made during this transaction permanent in the database.

  All data definition statements automatically perform a commit. For information, see the Side effects listing for each SQL statement.

  The COMMIT statement fails if the database server detects any invalid foreign keys. This behavior makes it impossible to end a transaction with any invalid foreign keys. Usually, foreign key integrity
is checked on each data manipulation operation. However, if the database option wait_for_commit is set On or a particular foreign key was defined with a CHECK ON COMMIT option, the database server delays integrity checking until the COMMIT statement is executed.

The use of COMMIT alone is equivalent to COMMIT WORK.

- **Syntax 2** You can use BEGIN TRANSACTION and COMMIT TRANSACTION statements in pairs to construct nested transactions. Nested transactions are similar to savepoints. When executed as the outermost of a set of nested transactions, the statement makes changes to the database permanent. When executed inside a transaction, the COMMIT TRANSACTION statement decreases the nesting level of transactions by one. When transactions are nested, only the outermost COMMIT makes the changes to the database permanent.

Syntax 2 is a Transact-SQL extension.

In Interactive SQL, you can also execute a COMMIT by:

- Pressing Ctrl+Shift+C.
- Clicking SQL » Commit.

In Interactive SQL, executing a COMMIT from the SQL menu or the keyboard shortcut does not modify the contents of the SQL Statements pane; however, the Results tab in the Results pane is cleared.

**Privileges**

None.

**Side effects**

Closes all cursors except those opened WITH HOLD.

Deletes all rows of declared temporary tables on this connection, unless they were declared using ON COMMIT PRESERVE ROWS.

If the database is not using a transaction log, each COMMIT operation causes an implicit checkpoint.

**See also**

- “wait_for_commit option” [SQL Anywhere Server - Database Administration]
- “auto_commit option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “commit_on_exit option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “Keyboard shortcuts for Interactive SQL” [SQL Anywhere Server - Database Administration]
- “SAVEPOINT statement” on page 954
- “List of connection properties” [SQL Anywhere Server - Database Administration]
- “BEGIN TRANSACTION statement [T-SQL]” on page 527
- “PREPARE TO COMMIT statement” on page 917
- “ROLLBACK statement” on page 950

**Standards and compatibility**

- **SQL/2008** Syntax 1 is a core feature. Syntax 2 is a Transact-SQL extension.
Example

The following statement commits the current transaction:

    COMMIT;

The following Transact-SQL batch reports successive values of @@trancount as 0, 1, 2, 1, 0.

    PRINT @@trancount
BEGIN TRANSACTION
    PRINT @@trancount
BEGIN TRANSACTION
    PRINT @@trancount
    COMMIT TRANSACTION
    PRINT @@trancount
    COMMIT TRANSACTION
    PRINT @@trancount
GO

CONFIGURE statement [Interactive SQL]

Opens the Interactive SQL Options window.

Syntax

CONFIGURE

Remarks

The CONFIGURE statement opens the Interactive SQL Options window. This window displays the current settings of all Interactive SQL options. It does not display or allow you to modify database options. You can configure Interactive SQL settings in this window.

Privileges

None.

Side effects

None.

See also

- “Customizing Interactive SQL” [SQL Anywhere Server - Database Administration]
- “SET OPTION statement” on page 972
- “Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

CONNECT statement [ESQL] [Interactive SQL]

Establishes a connection to a database.
Syntax 1 - Shared memory connections

CONNECT
  [ TO database-server-name ]
  [ DATABASE database-name ]
  [ AS connection-name ]
  [ USER ] userid [ IDENTIFIED BY password ]

  database-server-name, database-name, connection-name, userid, password :
  { identifier | string | hostvar }

Syntax 2 - TCP/IP connections

CONNECT USING connect-string

  connect-string : { identifier | string | hostvar }

Parameters

AS clause  A connection can optionally be named by specifying the AS clause. This allows multiple connections to the same database, or multiple connections to the same or different database servers, all simultaneously. Each connection has its own associated transaction. You may even get locking conflicts between your transactions if, for example, you try to modify the same record in the same database from two different connections.

For Syntax 2, a connect-string is a list of parameter settings of the form keyword=value, separated by semicolons, and must be enclosed in single quotes.

Remarks

The CONNECT statement establishes a connection to the database identified by database-name running on the database server identified by database-server-name. This statement is not supported in procedures, triggers, events, or batches.

Syntax 1 is only supported for shared memory connections to database servers running on the same computer. To connect to a local database server using TCP/IP or to a database server running on a different computer, use Syntax 2.

- Embedded SQL behavior  In embedded SQL, if no database-server-name is specified, the default local database server is assumed (the first database server started). If no database-name is specified, the first database on the given server is assumed.

  The WHENEVER statement, SET SQLCA, and some DECLARE statements do not generate code and may appear before the CONNECT statement in the source file. Otherwise, no statements are allowed until a successful CONNECT statement has been executed.

  The user ID and password are used for privilege checks on all dynamic SQL statements.

Note

For SQL Anywhere, only Syntax 1 is valid with embedded SQL. For UltraLite, both Syntax 1 and Syntax 2 can be used with embedded SQL.

- Interactive SQL behavior  If no database or server is specified in the CONNECT statement, Interactive SQL remains connected to the current database, rather than to the default server and
database. If a database name is specified without a server name, Interactive SQL attempts to connect to the specified database on the current server. If a server name is specified without a database name, Interactive SQL connects to the default database on the specified server.

For example, if the following batch is executed while connected to a database, the two tables are created in the same database.

```
CREATE TABLE t1 (c1 int);
CONNECT DBA IDENTIFIED BY sql;
CREATE TABLE t2 (c1 int);
```

No other database statements are allowed until a successful CONNECT statement has been executed.

When Interactive SQL is run in windowed mode, you are prompted for any missing connection parameters.

When Interactive SQL is running in command-prompt mode (-nogui is specified when you start Interactive SQL from a command line) or batch mode, or if you execute CONNECT without an AS clause, an unnamed connection is opened. If there is another unnamed connection already opened, the old one is automatically closed. Otherwise, existing connections are not closed when you execute a CONNECT statement.

Multiple connections are managed through the concept of a current connection. After a successful connect statement, the new connection becomes the current one. To switch to a different connection, use the SET CONNECTION statement. The DISCONNECT statement is used to drop connections.

When connecting to Interactive SQL, specifying CONNECT [ USER ] userid is the same as executing a SETUSER WITH OPTION userid statement.

In Interactive SQL, the connection information (including the database name, your user ID, and the database server) appears in the title bar above the SQL Statements pane. If you are not connected to a database, Not Connected appears in the title bar.

### Note
Both Syntax 1 and Syntax 2 are valid with Interactive SQL except that Interactive SQL does not support the hostvar argument.

This SQL statement is not supported for SAP HANA databases.

### Privileges
None.

### Side effects
None.
See also

- “GRANT CONNECT statement” on page 833
- “DISCONNECT statement [ESQL] [Interactive SQL]” on page 751
- “SET CONNECTION statement [Interactive SQL] [ESQL]” on page 966
- “SETUSER statement” on page 990
- “Troubleshooting: connections” [SQL Anywhere Server - Database Administration]
- “Connection parameters” [SQL Anywhere Server - Database Administration]
- “Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Syntax 1 is optional language feature F771 of the SQL/2008 standard. Syntax 2 is a vendor extension.
- Transact-SQL Both Syntax 1 and Syntax 2 are supported by Adaptive Server Enterprise.

Examples

The following are examples of CONNECT usage within embedded SQL.

```sql
EXEC SQL CONNECT AS :conn_name
USER :userid IDENTIFIED BY :password;
EXEC SQL CONNECT USER "DBA" IDENTIFIED BY "sql";
```

The following examples assume that the SQL Anywhere sample database has already been started.

Connect to a database from Interactive SQL. Interactive SQL prompts for a user ID and a password.

```sql
CONNECT;
```

Connect to the default database as user DBA from Interactive SQL. Interactive SQL prompts for a password.

```sql
CONNECT USER "DBA";
```

Connect to the sample database as user DBA from Interactive SQL.

```sql
CONNECT TO demo16 USER DBA IDENTIFIED BY sql;
```

Connect to the sample database using a connection string, from Interactive SQL.

```sql
CONNECT USING 'UID=DBA;PWD=sql;DBN=demo';
```

**CONTINUE statement**

Restarts a loop.

**Syntax**

```
CONTINUE [ statement-label ]
```
Remarks

The CONTINUE statement is a control statement that allows you to restart a loop. Execution continues at the first statement in the loop. When CONTINUE appears within a set of Watcom-SQL statements, *statement-label* must be specified.

When CONTINUE appears within a set of statements using Transact-SQL, *statement-label* must not be used.

Privileges

None.

Side effects

None.

See also

- “LOOP statement” on page 891
- “WHILE statement [T-SQL]” on page 1050
- “FOR statement” on page 805
- “BEGIN statement” on page 523
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 Vendor extension.
- Transact-SQL CONTINUE without a statement label is supported by Adaptive Server Enterprise.

Example

The following fragment shows how the CONTINUE statement is used to restart a loop. This example displays the odd numbers between 1 and 10.

```sql
BEGIN
    DECLARE i INT;
    SET i = 0;
    lbl:
        WHILE i < 10 LOOP
            SET i = i + 1;
            IF mod( i, 2 ) = 0 THEN
                CONTINUE lbl
            END IF;
            MESSAGE 'The value ' || i || ' is odd.' TO CLIENT;
        END LOOP lbl;
END
```

CREATE CERTIFICATE statement

Adds or replaces a certificate in the database from the given file or string. To create a certificate, use the Certificate Creation utility (createcert).
Syntax

CREATE [ OR REPLACE ] CERTIFICATE certificate-name  
FROM { certificate-string | variable-name | FILE file-name }

Parameters

FROM clause   This clause specifies a file, string, or variable containing a certificate.

Remarks

The CREATE CERTIFICATE statement adds or replaces a certificate in the database from the given file, string, or variable. The file, string, or variable should contain either a binary DER-format certificate or a text PEM-format certificate. DER-format certificates are converted and stored as PEM certificates.

Certificates that are stored in the database can be used by web service procedures and functions that make secure HTTPS connections to a web server. They can also be used to send secure messages using the xp_startsmtp system procedure.

When you add a certificate, it is added to the ISYSCERTIFICATE system table. Use the corresponding system view SYSCERTIFICATE to view the table.

The CREATE CERTIFICATE statement is not used to create an actual certificate. Use the Certificate Creation utility (createcert) to do this.

Privileges

You must have the MANAGE CERTIFICATES system privilege.

Side effects

Automatic commit.

See also

- “DROP CERTIFICATE statement” on page 752
- “CREATE PROCEDURE statement [Web service]” on page 628
- “CREATE FUNCTION statement [Web service]” on page 586
- “xp_startsmtp system procedure” on page 1341
- “SYSCERTIFICATE system view” on page 1350
- “sa_certificate_info system procedure” on page 1094
- “Certificate Creation utility (createcert)” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008   Vendor extension.

Example

The following example creates a certificate called mycert in the database using the contents of the specified certificate file.

```sql
CREATE CERTIFICATE mycert  
FROM FILE 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\Certificates\rsaroot.crt';
```
CREATE DATABASE statement

Creates a database.

Syntax

```
CREATE DATABASE db-filename-string [ create-option ... ]
```

create-option :

- `ACCENT { RESPECT | IGNORE | FRENCH }`
- `ASE [ COMPATIBLE ]`
- `BLANK PADDING { ON | OFF }`
- `CASE { RESPECT | IGNORE }`
- `CHECKSUM { ON | OFF }`
- `COLLATION collation-label ( collation-tailoring-string )`
- `DATABASE SIZE size { KB | MB | GB | PAGES | BYTES }`
- `DBA USER userid`
- `DBA PASSWORD password`
- `ENCODING encoding-label`
- `ENCRYPTED [ TABLE ] { algorithm-key-spec | OFF }`
- `JCONNECT { ON | OFF }`
- `MIRROR mirror-filename-string`
- `PAGE SIZE page-size`
- `NCHAR COLLATION nchar-collation-label ( collation-tailoring-string )`
- `SYSTEM PROCEDURE AS DEFINER { ON | OFF }`
- `TRANSACTION { LOG OFF | LOG ON [ log-filename-string ]}`

```
page-size :
2048 | 4096 | 8192 | 16384 | 32768
```

```
algorith-key-spec :
ON |
[ ON ] KEY key [ ALGORITHM AES-algorithm ]
[ ON ] ALGORITHM AES-algorithm KEY key
[ ON ] ALGORITHM 'SIMPLE'
```

```
AES-algorithm :
'AES' | 'AES256' | 'AES_FIPS' | 'AES256_FIPS'
```

```
key : quoted string
```

Parameters

`CREATE DATABASE` The file names (`db-filename-string`, `log-filename-string`, and `mirror-filename-string`) are strings containing operating system file names. As literal strings, they must be enclosed in single quotes.

**Note**

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

- If you specify a path, any backslash characters (\) must be doubled if they are followed by an n or an x. Escaping them prevents them from being interpreted as new line characters (\n) or as hexadecimal numbers (\x), according to the rules for strings in SQL.
Here are some examples where this is important.

```
CREATE DATABASE 'c:\temp\x41\x42\x43xyz.db'
DBA USER 'DBA' DBA PASSWORD 'sql';
```

The initial `\` sequence represents a backslash. The `\x` sequences represent the characters A, B, and C, respectively. The file name here is `ABCxyz.db`.

```
CREATE DATABASE 'c:\temp\nest.db'
DBA USER 'DBA' DBA PASSWORD 'sql';
```

To avoid having the `\n` sequence interpreted as a newline character, the backslash is doubled. See “Escape sequences” on page 8.

It is always safer to escape the backslash character. For example:

```
CREATE DATABASE 'c:\my_db.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
LOG ON 'e:\logdrive\my_db.log';
```

○ If you do not specify a path, or a relative path, the database file is created relative to the working directory of the database server. If you specify no path for a transaction log file, the file is created in the same directory as the database file. It is recommended that you store the database files and the transaction log on separate disks on the computer.

○ If you provide no file extension, a file is created with extension `.db` for databases, `.log` for the transaction log, and `.mlg` for the transaction log mirror.

○ The directory path is relative to the database server.

You cannot specify utility_db for `db-filename-string`. This name is reserved for the utility database.

**ACCENT clause** This clause is used to specify accent sensitivity for the database. Support for this clause is deprecated. Use the collation tailoring options provided for the COLLATION and NCHAR COLLATION clauses to specify accent sensitivity.

The ACCENT clause applies only when using the UCA (Unicode Collation Algorithm) for the collation specified in the COLLATION or NCHAR COLLATION clause. ACCENT RESPECT causes the UCA string comparison to respect accent differences between letters. For example, `e` is less than `é`. ACCENT FRENCH is similar to ACCENT RESPECT, except that accents are compared from right to left, consistent with the rules of the French language. ACCENT IGNORE causes string comparisons to ignore accents. For example, `e` is equal to `é`.

If accent sensitivity is not specified when the database is created, the default accent sensitivity for comparisons and sorting is insensitive, with one exception; for Japanese databases created with a UCA collation, the default accent sensitivity is sensitive.

**ASE COMPATIBLE clause** Do not create the SYS.SYSCOLUMNS and SYS.SYSINDEXES views. By default, these views are created for compatibility with system tables available in Watcom SQL (version 4 and earlier of this software). These views conflict with the Adaptive Server Enterprise compatibility views `dbo.syscolumns` and `dbo.sysindexes`.

**BLANK PADDING clause** SQL Anywhere compares all strings as if they are varying length and stored using the VARCHAR domain. This includes string comparisons involving fixed length CHAR or
NCHAR columns. In addition, SQL Anywhere never trims or pads values with trailing blanks when the values are stored in the database.

By default, SQL Anywhere treats blanks as significant characters. For example, the value 'a ' (the character 'a' followed by a blank) is not equivalent to the single-character string 'a'. Inequality comparisons also treat a blank as any other character in the collation.

If blank padding is enabled (specifying BLANK PADDING ON), the semantics of string comparisons more closely follow the ANSI/ISO SQL standard. With blank-padding enabled, SQL Anywhere ignores trailing blanks in any comparison.

In the example above, an equality comparison of 'a ' to 'a' in a blank-padded database returns TRUE. With a blank-padded database, fixed-length string values are padded with blanks when they are fetched by an application. Whether the application receives a string truncation warning on such an assignment is controlled by the ansi_blanks connection option.

**CASE clause** This clause is used to specify case sensitivity for the database. Support for this clause is deprecated. Use the collation tailoring options provided for the COLLATION and NCHAR COLLATION clauses to specify case sensitivity.

CASE RESPECT causes case-sensitive string comparisons for all CHAR and NCHAR data types. Comparisons using UCA consider the case of a letter only if the base letters and accents are all equal. For all other collations, uppercase and lowercase letters are distinct; for example, a is less than A, which is less than b, and so on. CASE IGNORE causes case-insensitive string comparisons. Uppercase and lowercase letters are considered to be exactly equal.

If case sensitivity is not specified when the database is created, default case sensitivity for comparisons and sorting is *insensitive*, with one exception; for Japanese databases created with a UCA collation, default case sensitivity is *sensitive*.

CASE RESPECT is provided for compatibility with the ISO/ANSI SQL standard. Identifiers in the database are always case insensitive, even in case-sensitive databases.

**CHECKSUM clause** Checksums are used to determine whether a database page has been modified on disk. When you create a database with global checksums enabled, a checksum is calculated for each page just before it is written to disk. The next time the page is read from disk, the page's checksum is recalculated and compared to the checksum stored on the page. If the checksums are different, then the page has been modified on disk and an error occurs. Databases created with global checksums enabled can also be validated using checksums. You can check whether a database was created with global checksums enabled by executing the following statement:

```sql
SELECT DB_PROPERTY ( 'Checksum' );
```

This query returns ON if global checksums are turned on, otherwise, it returns OFF. Global checksums are turned on by default, so if the CHECKSUM clause is omitted, ON is applied.

Regardless of the setting of this clause, the database server always enables write checksums for databases running on storage devices such as removable drives, and databases running on Windows Mobile to help provide early detection if the database file becomes corrupt. The database server also calculates checksums for critical pages during validation activities.
For databases that do not have global checksums enabled, you can enable write checksums by using the -wc options.

**COLLATION clause**  The collation specified by the COLLATION clause is used for sorting and comparison of character data types (CHAR, VARCHAR, and LONG VARCHAR). The collation provides character comparison and ordering information for the encoding (character set) being used. If the COLLATION clause is not specified, SQL Anywhere chooses a collation based on the operating system language and encoding.

The collation can be chosen from the list of collations that use the SQL Anywhere Collation Algorithm (SACA), or it can be the Unicode Collation Algorithm (UCA). If UCA is specified, you should also specify the ENCODING clause.

It is important to choose your collation carefully. It cannot be changed after the database has been created.

Optionally, you can specify collation tailoring options (collation-tailoring-string) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs, assembled in parentheses, following the collation name. For example, ... CHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)'.

**DATABASE SIZE clause**  Use this optional clause to set the initial size of the database file. You can use KB, MB, GB, or PAGES to specify units of kilobytes, megabytes, gigabytes, or pages respectively.

Specifying the file size at creation time is a way of preallocating space for the file. This helps reduce the risk of running out of space on the drive the database is located on. As well, it can help improve performance by increasing the amount of data that can be stored in the database before the database server needs to grow the database, which can be a time-consuming operation.

**DBA USER and DBA PASSWORD clauses**  Use these clauses to specify a DBA user ID and password for the database. If you do not specify these clauses, you will be prompted to specify the DBA user ID and password at creation time.

- User IDs cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons

- Passwords are case-sensitive and they cannot:
  - begin with white space, single quotes, or double quotes
  - end with white space
  - contain semicolons
  - be longer than 255 bytes in length

**ENCODING clause**  Most collations specified in the COLLATION clause dictate both the encoding (character set) and ordering. For those collations, the ENCODING clause should not be specified. However, if the value specified in the COLLATION clause is UCA (Unicode Collation Algorithm), use the ENCODING clause to specify a locale-specific encoding and get the benefits of the UCA for...
comparison and ordering. The ENCODING clause may specify UTF-8 or any single-byte encoding for
CHAR data types. ENCODING may not specify a multibyte encoding other than UTF-8.

If you choose the UCA collation, you can optionally specify collation tailoring options.

If COLLATION is set to UCA and ENCODING is not specified, then SQL Anywhere uses UTF-8.

**ENCRYPTED or ENCRYPTED TABLE clause** Encryption makes stored data unreadable. Use the
ENCRIPTED keyword (without TABLE) when you want to encrypt the entire database. Use the
ENCRYPTED TABLE clause when you only want to enable table encryption. Enabling table encryption
means that the tables that are subsequently created or altered using the ENCRYPTED clause are
encrypted using the settings you specified at database creation.

There are two levels of database and table encryption: simple and strong. Simple encryption is equivalent
to obfuscation. The data is unreadable, but someone with cryptographic expertise could decipher the data.
Strong encryption makes the data is unreadable and virtually undecipherable.

For simple encryption, specify ENCRYPTED ON ALGORITHM SIMPLE, or ENCRYPTED
ALGORITHM SIMPLE, or specify the ENCRYPTED ON clause without specifying an algorithm or key.

For strong encryption, specify ENCRYPTED ON ALGORITHM with a 128-bit or 256-bit AES
algorithm, and the KEY clause to specify an encryption key. It is recommended that you choose a value
for your key that is at least 16 characters long, contains a mix of uppercase and lowercase, and includes
numbers, letters, and special characters. A key can be specified as either a string or a variable name.

FIPS-certified encryption is not supported for use with Windows Mobile.

<table>
<thead>
<tr>
<th>Caution</th>
</tr>
</thead>
</table>
| For strongly encrypted databases, be sure to store a copy of the key in a safe location. If you lose the
 encryption key there is no way to access the data, even with the assistance of Technical Support. The
database must be discarded and you must create a new database. |

You can also create an encrypted copy of an existing database using the CREATE ENCRYPTED
DATABASE statement.

**JCONNECT clause** To allow the jConnect JDBC driver access to system catalog information, specify
JCONNECT ON. This clause installs the system objects that provide jConnect support. Specify
JCONNECT OFF to exclude the jConnect system objects. You can still use JDBC, as long as you do not
access system information. JCONNECT is ON by default.

**PAGE SIZE clause** The page size for a database can be 2048, 4096, 8192, 16384, or 32768 bytes. The
default page size is 4096 bytes. Large databases generally obtain performance benefits from a larger page
size, but there can be additional overhead associated with large page sizes.

For example:

```sql
CREATE DATABASE 'c:\temp\my_db.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
PAGE SIZE 4096;
```
**Note**
The page size cannot be larger than the page size used by the current server. The server page size is taken from the first set of databases started, or is set on the server command line using the -gp option.

**NCHAR COLLATION clause**
The collation specified by the NCHAR COLLATION clause is used for sorting and comparing national character data types (NCHAR, NVARCHAR, and LONG NVARCHAR). The collation provides character ordering information for the UTF-8 encoding (character set) used for national characters. If the NCHAR COLLATION clause is not specified, SQL Anywhere uses the Unicode Collation Algorithm (UCA). The only other allowed collation is UTF8BIN, which provides a binary ordering of all characters whose encoding is greater than 0x7E.

Optionally, you can specify collation tailoring options (collation-tailoring-string) for additional control over the sorting and comparing of characters. These options take the form of keyword=value pairs, assembled in a quoted string following the collation name. For example, ... NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)'. If you specify the ACCENT or CASE clause and a collation tailoring string that contains settings for case and accent, the values of the ACCENT and CASE clauses are used as defaults only.

**Note**
When you specify the UCA collation, all collation tailoring options are supported. For all other collations, only the case sensitivity tailoring option is supported.

Databases created with collation tailoring options cannot be started using a pre-10.0.1 database server.

**SYSTEM PROCEDURE AS DEFINER { ON | OFF } clause**
The SYSTEM PROCEDURE AS DEFINER clause specifies whether to execute pre-16.0 system procedures that perform privileged tasks with the privileges of the invoker or the definer (owner). ON means that these system procedures are executed with the privileges of the definer (owner). OFF means these system procedures are executed with the privileges of the invoker.

If this clause is not specified, the default is to run these procedures with the privileges of the invoker.

This setting does not impact user-defined procedures, or any system procedures introduced in version 16.0 or later. For information about what system procedures this affects, and impacts of the setting, see “Running pre-16.0 system procedures as invoker or definer” [SQL Anywhere Server - SQL Usage].

**TRANSACTION LOG clause**
The transaction log is a file where the database server logs all changes made to the database. The transaction log plays a key role in backup and recovery, and in data replication.

The MIRROR clause of the TRANSACTION clause allows you to provide a file name if you are using a transaction log mirror. A transaction log mirror is an identical copy of a transaction log, usually maintained on a separate device, for greater protection of your data. By default, SQL Anywhere does not use a transaction log mirror.

**Remarks**
Creates a database file with the supplied name and attributes. The database is stored as an operating system file. This statement is not supported in procedures, triggers, events, or batches.
You must be connected to a database to create another database. For example, connect to the utility database.

The account under which the database server is running must have write permissions on the directories where files are created.

Messages sent to the client indicate what type of database encryption is used for the database. If encryption is used, the algorithm being used is also displayed.

Privileges
Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects
An operating system file is created.

See also
- “ALTER DATABASE statement” on page 426
- “DROP DATABASE statement” on page 754
- “Initialization utility (dbinit)” [SQL Anywhere Server - Database Administration]
- “DatabaseKey (DBKEY) connection parameter” [SQL Anywhere Server - Database Administration]
- “The transaction log” [SQL Anywhere Server - Database Administration]
- “Validation utility (dbvalid)” [SQL Anywhere Server - Database Administration]
- “Corruption detection using checksums” [SQL Anywhere Server - Database Administration]
- “VALIDATE statement” on page 1045
- “CREATE ENCRYPTED DATABASE statement” on page 564
- “Table encryption” [SQL Anywhere Server - Database Administration]
- “The utility database” [SQL Anywhere Server - Database Administration]
- “ansi_blanks option” [SQL Anywhere Server - Database Administration]
- “-gp database server option” [SQL Anywhere Server - Database Administration]
- “-gu database server option” [SQL Anywhere Server - Database Administration]
- “-wc database server option” [SQL Anywhere Server - Database Administration]
- “-wc database option” [SQL Anywhere Server - Database Administration]
- “sa_validate system procedure” on page 1266
- “Collation tailoring options” [SQL Anywhere Server - Database Administration]
- “Collation considerations” [SQL Anywhere Server - Database Administration]
- “Alternate collations” [SQL Anywhere Server - Database Administration]
- “Simple encryption and strong encryption” [SQL Anywhere Server - Database Administration]
- “jConnect on Windows Mobile” [SQL Anywhere Server - Database Administration]
- “International languages and character sets” [SQL Anywhere Server - Database Administration]
- “Recommended character sets and collations” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.
- Transact-SQL The CREATE DATABASE statement is supported by Adaptive Server Enterprise, though with different clauses.
Examples

The following statement creates a database file named `temp.db` in the `C:\temp` directory:

```
CREATE DATABASE 'c:\temp\temp.db'
DBA USER 'DBA' DBA PASSWORD 'sql';
```

The following statement creates a database file named `mydb.db` in the `C:\temp` directory.

```
CREATE DATABASE 'C:\temp\mydb.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
TRANSACTION LOG ON
CASE IGNORE
PAGE SIZE 4096
BLANK PADDING OFF;
```

The following statement creates a database using code page 1252 and uses the UCA for both CHAR and NCHAR data types. Accents and case are respected during comparison and sorting.

```
CREATE DATABASE 'c:\temp\uca.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
COLLATION 'UCA'
ENCODING 'CP1252'
NCHAR COLLATION 'UCA'
ACCENT RESPECT
CASE RESPECT;
```

The following statement creates a database, `myencrypteddb.db`, that is encrypted using simple encryption:

```
CREATE DATABASE 'c:\temp\myencrypteddb.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
ENCRYPTED ON;
```

The following statement creates a database, `mystrongencryptdb.db`, that is encrypted using the key `gh67AB2` (strong encryption):

```
CREATE DATABASE 'c:\temp\mystrongencryptdb.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
ENCRYPTED ON KEY 'gh67AB2';
```

The following statement creates a database, `mytableencryptdb.db`, with table encryption enabled using simple encryption. Notice the keyword TABLE inserted after ENCRYPTED to indicate table encryption instead of database encryption:

```
CREATE DATABASE 'c:\temp\mytableencryptdb.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
ENCRYPTED TABLE ON;
```

The following statement creates a database, `mystrongencrypttabledb.db`, with table encryption enabled using simple encryption:

```
CREATE DATABASE 'c:\temp\mystrongencrypttabledb.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
ENCRYPTED TABLE ON 'SIMPLE';
```

The following statement creates a database file named `mydb.db` that uses collation 1252LATIN1. The NCHAR collation is set to UCA, with the locale set to es, and has case sensitivity and accent sensitivity enabled:
CREATE DATABASE 'c:\temp\my2.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
COLLATION '1252LATIN1(case=respect)'
NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)';

The following statement creates a database with a Greek collation:

CREATE DATABASE 'c:\temp\mydb.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
COLLATION '1253ELL';

The following statement creates a database named mydb.db with a transaction log mirror:

CREATE DATABASE 'c:\mydb.db'
DBA USER 'DBA' DBA PASSWORD 'sql'
TRANSACTION LOG ON 'mydb.log'
MIRROR 'd:\mydb.mlg';

CREATE DBSPACE statement

Defines a new database space and creates the associated database file.

Syntax

CREATE DBSPACE dbspace-name AS filename

Parameters

dbspace-name Specify a name for the dbspace. This is not the actual database file name, which you specify using filename. dbspace-name is an internal name you can refer to, for example in statements and procedures. You cannot use the following names for a dbspace because they are reserved for predefined dbspaces: system, temporary, temp, translog, and translogmirror.

An error is returned if you specify a value that contains a period (.)

filename Specify a name for the database file, including, optionally, the path to the file. If no path is specified, the database file is created in the same location (directory) as the main database file. If you specify a different location, the path is relative to the database server. The backslash (\) is an escape character in SQL strings, so each backslash must be doubled.

The filename parameter must be either a string literal or a variable.

Cloud note: For tenant databases in a cloud, when you specify the location of a dbspace, you can specify only a file name. You cannot specify a directory path.

Remarks

The CREATE DBSPACE statement creates a new database file. When a database is created, it is composed of one file. All tables and indexes created are placed in that file. CREATE DBSPACE adds a new file to the database. This file can be on a different disk drive than the main file, which means that the database can be larger than one physical device.
Note
If disk sandboxing is enabled, then the database’s operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

For each database, there is a limit of twelve dbspaces in addition to the main file.

Each object, such as a table or index, is contained entirely within one dbspace. The IN clause of the CREATE statement specifies the dbspace into which an object is placed. Objects are put into the system database file by default. You can also specify which dbspace tables are created in by setting the default_dbspace option before you create the tables.

Privileges
You must have the MANAGE ANY DBSPACE system privilege.

Side effects
Automatic commit. Automatic checkpoint.

See also
- “DROP DBSPACE statement” on page 755
- “default_dbspace option” [SQL Anywhere Server - Database Administration]
- “Strings” on page 6
- “Predefined dbspaces” [SQL Anywhere Server - Database Administration]
- “Additional dbspaces considerations” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example creates a dbspace called libbooks in the c:\directory. A subsequent CREATE TABLE statement creates a table called LibraryBooks in the libbooks dbspace.

```sql
CREATE DBSPACE libbooks
AS 'c:\library.db';
CREATE TABLE LibraryBooks (  
title char(100),
author char(50),
isbn char(30),
) IN libbooks;
```

CREATE DECRYPTED DATABASE statement

Creates a decrypted copy of an existing database, including all transaction logs and dbspaces.

Syntax

```
CREATE DECRYPTED DATABASE newfile  
FROM oldfile  
[ KEY key ]
```
Parameters

**FROM clause**  Use this clause to specify the name of the database to copy (*oldfile*).

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

**KEY clause**  Use this clause to specify the encryption key needed to decrypt the database. You can specify either a string or a variable name for the key. You do not specify the KEY clause if the existing database was encrypted with simple encryption, which does not require a key.

Remarks

The CREATE DECRYPTED DATABASE statement produces a new database file (*newfile*), and does not replace or remove the original database file (*oldfile*).

All encrypted tables in *oldfile* are not encrypted in *newfile*, and table encryption is not enabled.

**Note**
For databases created with SQL Anywhere 16 or later, the ISYSCOLSTAT, ISYSUSER, and ISYSEXTERNLOGIN system tables always remain encrypted to protect the data from unauthorized access.

If *oldfile* uses a transaction log or transaction log mirror, the files are renamed *newfile.log* and *newfile.mlg*, respectively.

If *oldfile* contains dbspace files, a D (decrypted) is added to the file name. For example, when you execute the CREATE DECRYPTED DATABASE statement, if *oldfile* is *mydbspace.dbs*, *newfile* becomes *mydbspace.dbsD*.

**Note**
If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

You cannot execute this statement on a database that requires recovery. This statement is not supported in procedures, triggers, events, or batches.

You cannot be connected to the database you are decrypting. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects

None.
CREATE DECRYPTED FILE statement

Creates a decrypted copy of a strongly encrypted database, and can be used to create decrypted copies of transaction logs, transaction log mirrors, and dbspaces.

Syntax

```
CREATE DECRYPTED FILE newfile
FROM oldfile KEY key
```

Parameters

- **FROM clause** Lists the file name of the encrypted file.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

- **KEY clause** Lists the key required to access the encrypted file. The key can be either a string or a variable name.

Remarks

Use this statement when your database requires recovery and you need to create a decrypted copy of the database for support reasons. You must also use this statement to decrypt any database-related files such as the transaction log, transaction log mirror, or dbspaces.

The original database file must be strongly encrypted using an encryption key. The resulting file is an exact copy of the encrypted file, without encryption and therefore requiring no encryption key.

Example

The first statement below creates an AES256-encrypted copy of the demo.db called demoEncrypted.db.

The second statement creates a decrypted copy of demoEncrypted.db called demoDecrypted.db.

```
CREATE ENCRYPTED DATABASE 'demoEncrypted.db'
FROM 'demo.db'
KEY 'Sd8f6654*Mnn'
ALGORITHM 'AES256';
CREATE DECRYPTED DATABASE 'demoDecrypted.db'
FROM 'demoEncrypted.db'
KEY 'Sd8f6654*Mnn';
```
Note
If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

If a database is decrypted using this statement, the corresponding transaction log file (and any dbspaces) must also be decrypted to use the database.

If a database requiring recovery is decrypted, its transaction log file must also be decrypted and recovery on the new database is necessary. The name of the transaction log file remains the same in this process, so if the database and transaction log file are renamed, then you need to run dblog -t on the resulting database.

You cannot use this statement on a database that has table encryption enabled. If you have tables you want to decrypt, use the NOT ENCRYPTED clause of the ALTER TABLE statements to decrypt them.

Note
For databases created with SQL Anywhere 16 or later, the ISYSCOLSTAT, ISYSUSER, and ISYSEXTERNLOGIN system tables always remain encrypted to protect the data from unauthorized access to the database file.

This statement is not supported in procedures, triggers, events, or batches.

You cannot be connected to the database you are decrypting. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

Privileges
Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects
None.

See also
- “ALTER TABLE statement” on page 486
- “CREATE ENCRYPTED FILE statement” on page 567
- “CREATE DECRYPTED DATABASE statement” on page 558
- “CREATE ENCRYPTED DATABASE statement” on page 564
- “-gu database server option” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example decrypts a fictitious encrypted database called encContacts, and creates a new unencrypted database called contacts.
CREATE DECRYPTED FILE 'contacts.db'  
FROM 'encContacts.db'  
KEY 'Sd8f6654*Mnn';

**CREATE DOMAIN statement**

Creates a domain in a database.

**Syntax**

```
CREATE { DOMAIN | DATATYPE } [ AS ] domain-name data-type  
[ [ NOT ] NULL ]  
[ DEFAULT default-value ]  
[ CHECK ( condition ) ]  
[ AS USER user-name ]
```

domain-name : identifier
data-type : built-in data type, with precision and scale

**Parameters**

**DOMAIN | DATATYPE clause**  
It is recommended that you use CREATE DOMAIN, rather than CREATE DATATYPE, because CREATE DOMAIN is defined in the SQL/2008 standard.

**NULL clause**  
This clause allows you to specify the nullability of a domain. When a domain is used to define a column, nullability is determined as follows:

- Nullability specified in the column definition.
- Nullability specified in the domain definition.
- If the nullability was not explicitly specified in either the column definition or the domain definition, then the setting of the allow_nulls_by_default option is used.

**CHECK clause**  
When creating a domain with a CHECK constraint, you can use a variable name prefixed with the @ sign in the CHECK constraint's search condition. When the data type is used in the definition of a column, such a variable is replaced by the column name. This allows a domain's CHECK constraint to be applied to each table column defined with that domain.

**AS USER clause**  
Specifies the owner of the object.

**Remarks**

Domains are aliases for built-in data types, including precision and scale values where applicable. They improve convenience and encourage consistency in the database.

Domains are objects within the database. Their names must conform to the rules for identifiers. Domain names are always case insensitive, as are built-in data type names.

The user who creates a data type is automatically made the owner of that data type. No owner can be specified in the CREATE DATATYPE statement. The domain name must be unique, and all users can access the data type without using the owner as prefix.
Domains can have CHECK conditions and DEFAULT values, and you can indicate whether the data type permits NULL values or not. These conditions and values are inherited by any column defined on the domain. Any conditions or values explicitly specified in the column definition override those specified for the domain.

**Privileges**

You must have the CREATE DATATYPE or CREATE ANY OBJECT system privilege to create domains owned by you. You cannot create domains owned by others.

**Side effects**

Automatic commit.

**See also**

- “allow_nulls_by_default option” [*SQL Anywhere Server - Database Administration*]
- “DROP DOMAIN statement” on page 756
- “SQL data types” on page 89

**Standards and compatibility**

- **SQL/2008** Domain support is optional SQL language feature F251 in the SQL/2008 standard.

**Examples**

The following statement creates a domain named address, which holds a 35-character string, and which may be NULL.

```
CREATE DOMAIN address CHAR(35) NULL;
```

The following statement creates a domain named ID, which does not allow NULLS, and which is set to autoincrement by default.

```
CREATE DOMAIN ID INT NOT NULL DEFAULT AUTOINCREMENT;
```

The following statement creates a domain named PhoneNumber, which uses a regular expression within a CHECK constraint to ensure that the string has a properly-formatted North American phone number of 12 characters, consisting of a 3-digit area code, 3-digit exchange, and 4-digit number separated by either dashes or a blank.

```
CREATE DOMAIN PhoneNumber CHAR(12) NULL CHECK( @PhoneNumber REGEXP '([2-9][0-9]{2}-[2-9][0-9]{2}-[0-9]{4})|([2-9][0-9]{2} [2-9][0-9]{2} [0-9]{4})');
```

Some columns in a database are used for employee names and others to store addresses. You might then define the following domains.

```
CREATE DOMAIN persons_name CHAR(30)
CREATE DOMAIN street_address CHAR(35);
```

Having defined these domains, you can use them much as you would the built-in data types. You can use these definitions to define a table, as follows. You need the CREATE TABLE privilege to execute the following statement.
CREATE TABLE myCustomers (  
  ID INT DEFAULT AUTOINCREMENT PRIMARY KEY,  
  Name persons_name,  
  Street street_address);  

In the above example, the table's primary key is specified to be of type integer. Indeed, many of your tables may require similar identifiers. Instead of specifying that these are integers, it is much more convenient to create an identifier domain for use in these applications.

When you create a domain, you can specify a default value and provide check constraint to ensure that no inappropriate values are typed into any column of this type.

Integer values are commonly used as table identifiers. A good choice for unique identifiers is to use positive integers. Since such identifiers are likely to be used in many tables, you could define the following domain.

CREATE DOMAIN identifier UNSIGNED INT  
  DEFAULT AUTOINCREMENT;

Using this definition, you can rewrite the definition of the Customers table, shown above.

CREATE TABLE Customers2 (  
  ID identifier PRIMARY KEY,  
  Name persons_name,  
  Street street_address  
);  

CREATE ENCRYPTED DATABASE statement

Creates an encrypted copy of an existing database, including all transaction logs and dbspaces.

Syntax 1 - Create an encrypted copy of a database

CREATE ENCRYPTED DATABASE newfile  
FROM oldfile  
[ KEY newkey ]  
[ ALGORITHM algorithm ]  
[ OLD KEY oldkey ]

algorithm:
  'SIMPLE'
  'AES'
  'AES256'
  'AES_FIPS'
  'AES256_FIPS'

Syntax 2 - Create a copy of a database with table encryption enabled

CREATE ENCRYPTED TABLE DATABASE newfile  
FROM oldfile  
[ KEY newkey ]  
[ ALGORITHM algorithm ]  
[ OLD KEY oldkey ]
Parameters

CREATE ENCRYPTED DATABASE clause  Use this clause to specify a name for the new encrypted database.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

CREATE ENCRYPTED TABLE DATABASE clause  Use this clause to specify a name for the new database. The new database is not encrypted, but has table encryption enabled.

FROM clause  Use this clause to specify the name of the original database (oldfile).

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

KEY clause  If algorithm-key is anything other than SIMPLE, use this clause to specify the encryption key for newfile. The key can be either a string or a variable name.

OLD KEY clause  Use this clause to specify the encryption key for oldfile. The key can be either a string or a variable name. This clause is only required if oldfile is encrypted with anything other than SIMPLE encryption.

ALGORITHM clause  Use this clause to specify the encryption algorithm to use for newfile. If you specify a KEY clause but do not specify the ALGORITHM clause, AES (128-bit encryption) is used by default. If you specify SIMPLE for algorithm, you do not specify a KEY clause.

Remarks

You can also use this statement to create a copy of a database and enable table encryption in the copy.

oldfile can be an unencrypted database, an encrypted database, or a database with table encryption enabled.

Syntax 1 takes an existing database, oldfile, and creates an encrypted copy of it, newfile.

Syntax 2 takes an existing database, oldfile, and creates a copy of it, newfile, with table encryption enabled. When you use this syntax, any tables encrypted in oldfile are encrypted in newfile as well. If no tables were encrypted in oldfile, but you want to encrypt them, you can execute an ALTER TABLE...ENCRYPTED statement on each table you want to encrypt.

Note

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

Neither syntax replaces or removes oldfile.

If oldfile uses transaction log or transaction log mirror files, they are renamed newfile.log and newfile.mlg respectively.
If `oldfile` contains dbspace files, an E (for encrypted) is added to the file name. For example, when you execute the CREATE ENCRYPTED DATABASE statement, the file `mydbspace.dbs` is changed to `mydbspace.dbsE`.

You can use this statement to change the encryption algorithm and key for a database. However, the CREATE ENCRYPTED DATABASE statement produces a new file (`newfile`), and does not replace or remove the previous version of the file (`oldfile`).

CREATE ENCRYPTED DATABASE and CREATE ENCRYPTED TABLE DATABASE cannot be executed against a database that requires recovery. These statements are not supported in procedures, triggers, events, or batches.

You cannot be connected to the database you are encrypting. You must be connected to a different database. For example, connect to the utility database. The database that you are encrypting must not be running.

You can also encrypt an existing database or change an existing encryption key by unloading and reloading the database using the `dbunload -an` option with either `-ek` or `-ep`.

You can also create an encrypted database, or a database with table encryption enabled, using the CREATE DATABASE statement.

**Note**
Not all platforms support FIPS-certified encryption. For a list of supported platforms, see [http://www.sybase.com/detail?id=1002288](http://www.sybase.com/detail?id=1002288).

**Privileges**
Your ability to execute this statement depends on the setting for the `-gu` database option, and whether you have the SERVER OPERATOR system privilege.

**Side effects**
None.

**See also**
- “Tips on rebuilding databases using the Unload utility” ([SQL Anywhere Server - SQL Usage](#))
- “Database encryption and decryption” ([SQL Anywhere Server - Database Administration](#))
- “Table encryption” ([SQL Anywhere Server - Database Administration](#))
- “Simple encryption and strong encryption” ([SQL Anywhere Server - Database Administration](#))
- “CREATE DECRYPTED DATABASE statement” on page 558
- “CREATE ENCRYPTED FILE statement” on page 567
- “CREATE DECRYPTED FILE statement” on page 560
- “CREATE DATABASE statement” on page 549
- “ALTER TABLE statement” on page 486
- “Initialization utility (dbinit)” ([SQL Anywhere Server - Database Administration](#))
- “-gu database server option” ([SQL Anywhere Server - Database Administration](#))
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example creates an encrypted copy of the sample database called `demoEnc.db`. The new database is encrypted with AES256 encryption.

```sql
CREATE ENCRYPTED DATABASE 'demoEnc.db'
FROM 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.db'
KEY 'Sd8f6654*Mnn'
ALGORITHM 'AES256';
```

The following example creates a copy of the sample database called `demoTableEnc.db`. Table encryption is enabled on the new database. Since a key was specified with no algorithm, AES encryption is used.

```sql
CREATE ENCRYPTED TABLE DATABASE 'demoTableEnc.db'
FROM 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.db'
KEY 'Sd8f6654';
```

**CREATE ENCRYPTED FILE statement**

[This topic has been updated for build 1823.]

Creates a strongly encrypted copy of a database file, transaction log, transaction log mirror, or dbspace.

**Syntax**

```sql
CREATE ENCRYPTED FILE  newfile
FROM  oldfile
{ KEY key | KEY key OLD KEY oldkey }
[ ALGORITHM { 'AES'
 | 'AES256'
 | 'AES_FIPS'
 | 'AES256_FIPS' } ]
```

**Parameters**

**FROM clause**  Specifies the name of the existing file (`oldfile`) on which to execute the CREATE ENCRYPTED FILE statement.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

**KEY clause**  Specifies the encryption key to use. The key can be either a string or a variable name.

**OLD KEY clause**  Specifies the current key with which the file is encrypted. The key can be either a string or a variable name.

**ALGORITHM clause**  Specifies the algorithm used to encrypt the file. If you do not specify an algorithm, AES (128-bit encryption) is used by default.
Remarks

Use this statement when your database requires recovery and you need to create an encrypted copy of the
database for support reasons. You must also use this statement to encrypt any database-related files such
as the transaction log, transaction log mirror, or dbspace files.

You cannot be connected to the database you are creating the encrypted file for. You must be connected
to a different database. For example, connect to the utility database. The database that you are encrypting
must not be running.

When encrypting the database-related files, you must specify the same algorithm and key for all files
related to the database.

If oldfile has dbspaces or transaction log files associated with it and you encrypt those too, you must
ensure that the new name and location of those files is stored with the new database. To do so:

- run dblog -t on the new database to change the name and location of the transaction log
- run dblog -m on the new database to change the name and location of the transaction log mirror
- execute an ALTER DBSPACE statement on the new database to change the location and name of the
dbspace files

Note
If disk sandboxing is enabled, then the database's operations are limited to the directory where the main
database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

You can use this statement to change the encryption algorithm and key for a database. However, the
CREATE ENCRYPTED FILE statement produces a new file (newfile), and does not replace or remove
the previous version of the file (oldfile).

The name of the transaction log file remains the same in this process, so if the database and transaction
log file are renamed, then you need to run dblog -t on the resulting database.

You can also encrypt an existing database or change an existing encryption key by unloading and
reloading the database using the dbunload -an option with either -ek or -ep.

If you have a database on which table encryption is enabled, you cannot encrypt the database using this
statement. However, you can use this statement to change the key used for table encryption. To encrypt a
database that has table encryption enabled, use the CREATE ENCRYPTED DATABASE statement.

FIPS-certified encryption is not supported for use with Windows Mobile.

This statement is not supported in procedures, triggers, events, or batches.

Note
Not all platforms support FIPS-certified encryption. For a list of supported platforms, see http://
Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects

None.

See also

- “CREATE ENCRYPTED DATABASE statement” on page 564
- “Database encryption and decryption” [SQL Anywhere Server - Database Administration]
- “CREATE ENCRYPTED DATABASE statement” on page 564
- “CREATE DECRYPTED FILE statement” on page 560
- “CREATE DECRYPTED DATABASE statement” on page 558
- “Unload utility (dbunload)” [SQL Anywhere Server - Database Administration]
- “Transaction Log utility (dblog)” [SQL Anywhere Server - Database Administration]
- “-gu database server option” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example encrypts the sample database demo.db and creates a new database called demo2.db that is encrypted with AES_FIPS encryption. The new database file is placed in the server's current working directory.

```
CREATE ENCRYPTED FILE 'demo2.db'
FROM 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.db'
KEY 'Sd8f6654*Mnn'
ALGORITHM 'AES_FIPS';
```

The following example encrypts the sample database demo.db and its transaction log file demo.log. The new database and log files are placed in the server's current working directory. You must run dblog - ek Sd8f6654*Mnn -t demo3.log demo3.db, since the new database file demo3.db still points to the old log file.

```
CREATE ENCRYPTED FILE 'demo3.db'
FROM 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.db'
KEY 'Sd8f6654*Mnn';
CREATE ENCRYPTED FILE 'demo3.log'
FROM 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.log'
KEY 'Sd8f6654*Mnn';
```

To change the encryption key for a database, first create a copy of the database file using the new key, as shown in this statement:

```
CREATE ENCRYPTED FILE 'newdemo.db'
FROM 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.db'
KEY 'newkey' OLD KEY 'oldkey';
```

The new database file is placed in the server's current working directory. Once you have created the encrypted file, delete demo.db, move newdemo.db to the same directory as the old file, and then rename it to be demo.db.
CREATE EVENT statement

Defines an event and its associated handler for automating predefined actions, and to define scheduled actions.

Syntax

```
CREATE EVENT [ owner.]event-name
[ TYPE event-type ]
[ WHERE trigger-condition [ AND trigger-condition ] ... ]
[ SCHEDULE schedule-spec, ... ]
[ ENABLE | DISABLE ]
[ AT { CONSOLIDATED | REMOTE | ALL } ]
[ FOR { PRIMARY | ALL } ]
[ HANDLER BEGIN ...
... END ]
```

event-type :
- BackupEnd
- Connect
- ConnectFailed
- DatabaseStart
- DBDiskSpace
- Deadlock
- "Disconnect"
- GlobalAutoincrement
- GrowDB
- GrowLog
- GrowTemp
- LogDiskSpace
- MirrorFailover
- MirrorServerDisconnect
- RAISERROR
- ServerIdle
- TempDiskSpace

trigger-condition :
- event_condition( condition-name ) {
  =
  <
  >
  !=
  <=
  >=
} value

schedule-spec :
- [ schedule-name ]
  { START TIME start-time | BETWEEN start-time AND end-time }
  { EVERY period { HOURS | MINUTES | SECONDS } }
  { ON { ( day-of-week, ... ) | ( day-of-month, ... ) } }
  { START DATE start-date }

event-name : identifier
schedule-name : identifier

day-of-week : string

day-of-month : integer

value : integer

period : integer

start-time : time

end-time : time

start-date : date

Parameters

**CREATE EVENT clause**  The event name is an identifier. An event has a creator, which is the user creating the event, and the event handler executes with the privileges of that creator. This is the same as stored procedure execution. You cannot create events owned by other users.

**TYPE clause**  You can specify the TYPE clause with an optional WHERE clause, or specify the SCHEDULE.

The event-type is one of the listed set of system-defined event types. The event types are case insensitive. To specify the conditions under which this event-type triggers the event, use the WHERE clause.

- **DiskSpace event types**  If the database contains an event handler for one of the DiskSpace types, the database server checks the available space on each device associated with the relevant file every 30 seconds.

  In the event the database has more than one dbspace, on separate drives, DBDiskSpace checks each drive and acts depending on the lowest available space.

  The LogDiskSpace event type checks the location of the transaction log and any transaction log mirror, and reports based on the least available space.

  Disk space event types are not supported on Windows Mobile.

  The TempDiskSpace event type checks the amount of temporary disk space.

  If the appropriate event handlers have been defined (DBDiskSpace, LogDiskSpace, or TempDiskSpace), the database server checks the available space on each device associated with a database file every 30 seconds. Similarly, if an event has been defined to handle the system event type ServerIdle, the database server notifies the handler when no requests have been processed during the previous 30 seconds.

  You can specify the -fc option when starting the database server to implement a callback function when the database server encounters a file system full condition.

- **GlobalAutoincrement event type**  The event fires on each insert when the number of remaining values for a GLOBAL AUTOINCREMENT is less than 1% of the end of its range. A typical action
for the handler could be to request a new value for the global_database_id option, based on the table and number of remaining values which are supplied as parameters to this event.

You can use the event_condition function with RemainingValues as an argument for this event type.

- **ServerIdle event type**  If the database contains an event handler for the ServerIdle type, the database server checks for server activity every 30 seconds.

- **Database mirroring event types**  The MirrorServerDisconnect event fires when a connection from the primary database server to the mirror server or arbiter server is lost, and the MirrorFailover event fires whenever a server takes ownership of the database.

**WHERE clause**  The trigger condition determines the condition under which an event is fired. For example, to take an action when the disk containing the transaction log becomes more than 80% full, use the following triggering condition:

```sql
... WHERE event_condition( 'LogFreePercent' ) < 20 ...
```

The argument to the event_condition function must be valid for the event type.

You can use multiple AND conditions to make up the WHERE clause, but you cannot use OR conditions or other conditions.

**SCHEDULE clause**  This clause specifies when scheduled actions are to take place. The sequence of times acts as a set of triggering conditions for the associated actions defined in the event handler.

You can create more than one schedule for a given event and its associated handler. This permits complex schedules to be implemented. You must provide a schedule-name when there is more than one schedule; the schedule-name is optional if you provide only a single schedule.

A scheduled event is recurring if its definition includes EVERY or ON; if neither of these reserved words is used, the event executes at most once. An attempt to create a non-recurring scheduled event for which the start time has passed generates an error. When a non-recurring scheduled event has passed, its schedule is deleted, but the event handler is not deleted.

Scheduled event times are calculated when the schedules are created, and again when the event handler completes execution. The next event time is computed by inspecting the schedule or schedules for the event, and finding the next schedule time that is in the future. If an event handler is instructed to run every hour between 9:00 and 5:00, and it takes 65 minutes to execute, it runs at 9:00, 11:00, 1:00, 3:00, and 5:00. If you want execution to overlap, you must create more than one event.

The subclauses of a schedule definition are as follows:

- **START TIME clause**  The first scheduled time for each day on which the event is scheduled. The start-time parameter is a string, and cannot be a variable or an expression such as NOW(). If a START DATE is specified, the START TIME refers to that date and each subsequent day (if the schedule includes EVERY or ON). If no START DATE is specified, the START TIME is on the current day (unless the time has passed) and each subsequent day (if the schedule includes EVERY or ON). The clause START TIME start-time is equivalent to BETWEEN start-time AND '23:59:59'.
● **BETWEEN...AND clause**  A range of times during the day outside which no scheduled times occur. The start-time and end-time parameters are strings, and cannot be variables or expressions such as NOW(). If a START DATE is specified, the scheduled times do not occur until that date.

● **EVERY clause**  An interval between successive scheduled events. Scheduled events occur only after the START TIME for the day, or in the range specified by BETWEEN...AND.

● **ON clause**  A list of days on which the scheduled events occur. The default is every day if EVERY is specified. Days can be specified as days of the week or days of the month.

Days of the week are Mon, Tues, and so on. You may also use the full forms of the day, such as Monday. You must use the full forms of the day names if the language you are using is not English, is not the language requested by the client in the connection string, and is not the language which appears in the database server messages window.

Days of the month are integers from 0 to 31. A value of 0 represents the last day of any month.

● **START DATE clause**  The date on which scheduled events are to start occurring. This value is a string, and cannot be a variable or an expression such as TODAY(). The default is the current date.

Each time a scheduled event handler is completed, the following actions are taken to calculate the next scheduled time and date for the event:

1. If the EVERY clause is used, find whether the next scheduled time falls on the current day, and is before the end time specified by the BETWEEN...AND clause, if it was specified. If so, that is the next scheduled time.

2. If the next scheduled time does not fall on the current day, find the next date on which the event is to be executed and use the START TIME for that date, or the beginning of the BETWEEN...AND range.

● **ENABLE | DISABLE clause**  By default, event handlers are enabled. When DISABLE is specified, the event handler does not execute even when the scheduled time or triggering condition occurs. A TRIGGER EVENT statement does not cause a disabled event handler to be executed.

● **AT clause**  This clause should be used only in the following circumstance: in a SQL Remote setup, use the AT clause against your remote or consolidated databases to restrict the databases at which the event is handled.

If you do not use the AT clause when creating events for SQL Remote, all databases execute the event. When executed on a consolidated database, this statement does not affect remote databases that have already been extracted.

● **FOR clause**  This clause should only be used in the following circumstance: in a database mirroring or read-only scale-out system, use the FOR clause to restrict the databases at which the event is handled.

If you do not use the FOR clause when creating an event for a database in a mirroring or read-only scale-out system, then only the database that is running on the primary server executes the event. The following subclauses are supported:
○ **FOR PRIMARY**  The event executes only on the server currently acting as the primary server. The default is the PRIMARY sub clause.

When the FOR PRIMARY clause (or the FOR clause is not specified) is used with the DatabaseStart event type, the event executes when a server becomes the primary server for the database.

○ **FOR ALL**  The event executes on all servers in the system.

When the FOR ALL clause is used with the DatabaseStart event type, the event is executed when any database starts. If in a mirroring system the event did not run when the database started (for example, the database was running before the event was created), then the event can execute during a fail over. For example you start a database mirroring system, you create a DatabaseStart event with the FOR ALL clause, and then you stop the primary server, which causes a fail over. In this example, the event executes on the new primary server. The DatabaseStart event will not execute during subsequent fail overs.

**• HANDLER clause**  Each event has one handler.

**Remarks**

Events can be used for:

• **Scheduling actions**  The database server executes actions on a timed schedule. You can use this capability to complete scheduled tasks such as backups, validity checks, and queries used to add data to reporting tables.

• **Event handling actions**  The database server executes actions when a predefined event occurs. You can use this capability to complete scheduled tasks such as restrict disk space when a disk fills beyond a specified percentage. Event handler actions are committed if errors are not detected during execution, and rolled back if errors are detected.

An event definition includes two distinct pieces. The trigger condition can be an occurrence, such as a disk filling up beyond a defined threshold. A schedule is a set of times, each of which acts as a trigger condition. When a trigger condition is satisfied, the event handler executes. The event handler includes one or more actions specified inside a compound statement (BEGIN... END).

If no trigger condition or schedule specification is supplied, only an explicit TRIGGER EVENT statement can trigger the event. During development, you may want to test event handlers using TRIGGER EVENT, and add the schedule or WHERE clause once testing is complete.

Event errors are logged to the database server message log.

After each execution of an event handler, a COMMIT occurs if no errors occurred. A ROLLBACK occurs if there was an error.

When event handlers are triggered, the database server makes context information, such as the connection ID that caused the event to be triggered, available to the event handler using the event_parameter function.
Event handlers execute on a separate connection, but the separate connection does not count towards the ten-connection limit of the personal database server.

Privileges

You must have the MANAGE ANY EVENT or CREATE ANY OBJECT system privilege.

Side effects

Automatic commit.

See also

- “BEGIN statement” on page 523
- “ALTER EVENT statement” on page 436
- “COMMENT statement” on page 538
- “DROP EVENT statement” on page 757
- “TRIGGER EVENT statement” on page 1018
- “EVENT_PARAMETER function [System]” on page 240
- “EVENT_CONDITION function [System]” on page 238
- “-fc database server option” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]
- “Trigger conditions for events” [SQL Anywhere Server - Database Administration]
- “Database server action logging” [SQL Anywhere Server - Database Administration]
- “Using database mirroring system events to send notification email when failover occurs” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Examples

Instruct the database server to perform an automatic backup to tape using the first tape drive, every day at 1 A.M.

```
CREATE EVENT DailyBackup
SCHEDULE daily_backup
START TIME '1:00AM' EVERY 24 HOURS
HANDLER
    BEGIN
        BACKUP DATABASE TO '\\.\tape0'
        ATTENDED OFF
    END;
```

Instruct the database server to perform an incremental backup daily at 1:00 A.M.

```
CREATE EVENT IncrementalBackup
SCHEDULE
    START TIME '1:00 AM' EVERY 24 HOURS
HANDLER
    BEGIN
        BACKUP DATABASE DIRECTORY 'c:\backup'
        TRANSACTION LOG ONLY
        TRANSACTION LOG RENAME MATCH
    END;
```
Instruct the database server to perform an automatic backup of the transaction log only, every hour, Monday to Friday between 8 A.M. and 6 P.M.

```sql
CREATE EVENT HourlyLogBackup
SCHEDULE hourly_log_backup
BETWEEN '8:00AM' AND '6:00PM'
EVERY 1 HOURS ON
('Monday','Tuesday','Wednesday','Thursday','Friday')
HANDLER
BEGIN
  BACKUP DATABASE DIRECTORY 'c:\database\backup'
  TRANSACTION LOG ONLY
  TRANSACTION LOG RENAME
END;
```

Determine when an event is next scheduled to run:

```sql
SELECT DB_EXTENDED_PROPERTY( 'NextScheduleTime', 'HourlyLogBackup');
```

Create an event that can run on any server in a mirroring or read-only scale-out system.

```sql
CREATE EVENT evt1 FOR ALL
HANDLER
BEGIN
  MESSAGE CURRENT TIME || ' evt1 active' TO CONSOLE;
END
```

**CREATE EXISTING TABLE statement**

Creates a new proxy table, which represents an existing object on a remote server.

**Syntax**

```sql
CREATE EXISTING TABLE [owner.]table-name
  [ column-definition, ... ]
AT location-string

column-definition :
  column-name data-type NOT NULL

location-string :
  remote-server-name.[db-name].[owner].object-name
| remote-server-name[db-name][owner].object-name
```

**Parameters**

**AT clause** The AT clause specifies the location of the remote object. The AT clause supports the semicolon (;) as a delimiter. If a semicolon is present anywhere in the `location-string` string, the semicolon is the field delimiter. If no semicolon is present, a period is the field delimiter. This behavior allows file names and extensions to be used in the database and owner fields.

The string in the AT clause can also contain local or global variable names enclosed in braces (`{variable-name}`). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains `'access;{myfile};a1'` indicates that `@myfile` is a SQL variable and that the current contents of the `@myfile` variable should be substituted when the proxy table is created.
Remarks

The CREATE EXISTING TABLE statement creates a new, local, proxy table that maps to a table at an external location. The CREATE EXISTING TABLE statement is a variant of the CREATE TABLE statement. The EXISTING keyword is used with CREATE TABLE to specify that a table already exists remotely and to import its metadata. This establishes the remote table as a visible entity to SQL Anywhere users. The software verifies that the table exists at the external location before it creates the table.

If the object does not exist (either as a host data file or remote server object), the statement is rejected with an error message.

Index information from the host data file or remote server table is extracted and used to create rows for the ISYSIDX system table. This information defines indexes and keys in server terms and enables the query optimizer to consider any indexes that may exist on this table.

Referential constraints are passed to the remote location when appropriate.

If columndefinitions are not specified, SQL Anywhere derives the column list from the metadata it obtains from the remote table. If columndefinitions are specified, SQL Anywhere verifies the columndefinitions. Column names, data types, lengths, the identity property, and null properties are checked for the following:

- Column names must match identically (although case is ignored).
- Data types in the CREATE EXISTING TABLE statement must match or be convertible to the data types of the column on the remote location. For example, a local column data type is defined as money, while the remote column data type is numeric.
- Each column's NULL property is checked. If the local column's NULL property is not identical to the remote column's NULL property, a warning message is issued, but the statement is not aborted.
- Each column's length is checked. If the length of CHAR, VARCHAR, BINARY, VARBINARY, DECIMAL and/or NUMERIC columns do not match, a warning message is issued, but the command is not aborted.

You may choose to include only a subset of the actual remote column list in your CREATE EXISTING statement.

This statement is not supported on Windows Mobile.

Privileges

You must have the CREATE PROXY TABLE system privilege to create proxy tables owned by you. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create proxy tables owned by others.

Side Effects

Automatic commit.
See also

- “CREATE TABLE statement” on page 690
- “Proxy table locations” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- **SQL/2008**  Vendor extension.
- **Transact-SQL**  Supported by Adaptive Server Enterprise. The format of location-string is implementation-defined.

Examples

Create a proxy table named blurbs for the blurbs table at the remote server server_a.

```sql
CREATE EXISTING TABLE blurbs
( author_id ID not null,
  copy text not null)
AT 'server_a.db1.joe.blurbs';
```

Create a proxy table named blurbs for the blurbs table at the remote server server_a. The database server derives the column list from the metadata it obtains from the remote table.

```sql
CREATE EXISTING TABLE blurbs
AT 'server_a.db1.joe.blurbs';
```

Create a proxy table named rda_employees for the Employees table at the remote server rda.

```sql
CREATE EXISTING TABLE rda_employees
AT 'rda...Employees';
```

Create a proxy table named rda_employees for a table that is specified by the SQL variable table_name at the remote server rda.

```sql
CREATE EXISTING TABLE rda_employees
AT 'rda...(table_name)';
```

**CREATE EXTERNLOGIN statement**

Assigns an alternate login name and password to be used when communicating with a remote server.

**Syntax**

```sql
CREATE EXTERNLOGIN login-name
TO remote-server
[ REMOTE LOGIN remote-user [ IDENTIFIED BY remote-password ] ]
```

**Parameters**

- **login-name**  specifies the local user login name. When using integrated logins, the login-name is the database user to which the Windows user or group is mapped.

- **TO clause**  The TO clause specifies the name of the remote server.
**REMOTE LOGIN clause**  The REMOTE LOGIN clause specifies the user account on *remote-server* for the local user *login-name*. Values for the REMOTE LOGIN clause are restricted to 128 bytes.

**IDENTIFIED BY clause**  The IDENTIFIED BY clause specifies the *remote-password* for *remote-user*. The *remote-user* and *remote-password* combination must be valid on the remote-server.

If you omit the IDENTIFIED BY clause, the password is sent to the remote server as NULL. However, if you specify IDENTIFIED BY "" (an empty string), then the password sent is the empty string.

**Remarks**

By default, SQL Anywhere uses the names and passwords of its clients whenever it connects to a remote server on behalf of those clients. CREATE EXTERNLOGIN assigns an alternate login name and password to be used when communicating with a remote server.

The REMOTE LOGIN clause is required only when the remote server requires a user ID and password for the connection. Having an external login without a remote login allows the DBA to control who can access the remote server and tells the remote access layer that logging in to the remote server does not require a user ID and password. For example, the directory access server class requires an external login for restricting access to the directory server, but remote login is not needed because the directory server does not perform user ID and password validation.

The password is stored internally in encrypted form. The *remote-server* must be known to the local server by an entry in the ISYSSERVER table.

Sites with automatic password expiration should plan for periodic updates of passwords for external logins.

CREATE EXTERNLOGIN cannot be used from within a transaction.

This statement is not supported on Windows Mobile.

**Privileges**

You must have the MANAGE ANY USER system privilege.

**Side effects**

Automatic commit.

**See also**

- “DROP EXTERNLOGIN statement” on page 758
- “CREATE SERVER statement” on page 657
- “Creating external logins (Sybase Central)” [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

This fictitious example maps a local user, DBA, to user sa with password Plankton when connecting to the server sybase1.
CREATE EXTERNLOGIN DBA
TO sybase1
REMOTE LOGIN sa
IDENTIFIED BY Plankton;

CREATE FUNCTION statement [External call]

Creates an interface to a native or external function.

Syntax

```
CREATE [ OR REPLACE ] FUNCTION [ owner.]function-name
  ( [ parameter, ... ] )
  [ SQL SECURITY { INVOKER | DEFINER } ]
  RETURNS data-type
  [ [ NOT ] DETERMINISTIC ]
  { EXTERNAL NAME 'native-call'
    | EXTERNAL NAME 'c-call' LANGUAGE { C_ESQL32 | C_ESQL64 | C_ODBC32 | C_ODBC64 }
    | EXTERNAL NAME 'clr-call' LANGUAGE CLR
    | EXTERNAL NAME 'perl-call' LANGUAGE PERL
    | EXTERNAL NAME 'php-call' LANGUAGE PHP
    | EXTERNAL NAME 'java-call' LANGUAGE JAVA }

parameter:
  [ IN ] parameter-name data-type [ DEFAULT expression ]

native-call:
  [ operating-system:]function-name@library

result-column:
  column-name data-type

c-call:
  [ operating-system:]function-name@library; ... 

clr-call:
  dll-name::function-name( param-type-1[, ... ] )

perl-call:
  <file=perl-file> $sa_perl_return = perl-subroutine( $sa_perl_arg0[, ... ] )

php-call:
  <file=php-file> print php-func( $argv[1][, ... ] )

java-call:
  [package-name,]class-name.method-name method-signature

operating-system:
  Unix

method-signature:
  ( [ field-descriptor, ... ] ) return-descriptor

field-descriptor and return-descriptor:
  { Z
```
Parameters

Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type, and must be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the function. However, function parameters are IN by default.

When functions are executed, not all parameters need to be specified. If a DEFAULT value is provided in the CREATE FUNCTION statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, an error is given.

OR REPLACE clause Specifying OR REPLACE creates a new function, or replaces an existing function with the same name. This clause changes the definition of the function, but preserves existing privileges. You cannot use the OR REPLACE clause with temporary functions.

RETURNS clause Use the RETURNS clause to specify the data type for the result of the function.

SQL SECURITY clause The SQL SECURITY clause defines whether the function is executed as the INVOKER (the user who is calling the function), or as the DEFINER (the user who owns the function). The default is DEFINER. For external calls, this clause establishes the ownership context for unqualified object references in the external environment.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the function. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, care should be taken to qualify all object names (tables, procedures, and so on) with their appropriate owner. For example, suppose user1 creates the following function:

```sql
CREATE FUNCTION user1.myFunc()
RETURNS INT
SQL SECURITY INVOKER
BEGIN
DECLARE res INT;
SELECT COUNT(*) INTO res FROM table1;
RETURN res;
END;
```

If user2 attempts to run this function and a table user2.table1 does not exist, a table lookup error results. Additionally, if a user2.table1 does exist, that table is used instead of the intended user1.table1. To prevent this situation, qualify the table reference in the statement (user1.table1, instead of just table1).

[ NOT ] DETERMINISTIC clause Use this clause to indicate whether functions are deterministic or non-deterministic. If this clause is omitted, then the deterministic behavior of the function is unspecified (the default).
If a function is declared as DETERMINISTIC, it should return the same value every time it is invoked with the same set of parameters.

If a function is declared as NOT DETERMINISTIC, then it is not guaranteed to return the same value for the same set of parameters. A function declared as NOT DETERMINISTIC is re-evaluated each time it is called in a query. This clause must be used when it is known that the function result for a given set of parameters can vary.

Also, functions that have side effects such as modifying the underlying data should be declared as NOT DETERMINISTIC. For example, a function that generates primary key values and is used in an INSERT...SELECT statement should be declared NOT DETERMINISTIC:

```sql
CREATE FUNCTION keygen( increment INTEGER )
RETURNS INTEGER
NOT DETERMINISTIC
BEGIN
    DECLARE keyval INTEGER;
    UPDATE counter SET x = x + increment;
    SELECT counter.x INTO keyval FROM counter;
    RETURN keyval
END
INSERT INTO new_table
SELECT keygen(1), ...
FROM old_table;
```

Functions can be declared as DETERMINISTIC if they always return the same value for given input parameters.

**EXTERNAL NAME native-call clause**

```
native-call :
[operating-system:]function-name@library; ...
```

**Note**
The EXTERNAL NAME clause is not supported for TEMPORARY functions.

A function using the EXTERNAL NAME clause with no LANGUAGE attribute defines an interface to a native function written in a programming language such as C. The native function is loaded by the database server into its address space.

The `library` name can include the file extension, which is typically `.dll` on Windows and `.so` on Unix. In the absence of the extension, the software appends the platform-specific default file extension for libraries. For example:

```sql
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib.dll;Unix:mystring@mylib.so';
```

A simpler way to write the above EXTERNAL NAME clause, using platform-specific defaults, is as follows:
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib';

When called, the library containing the function is loaded into the address space of the database server. The native function will execute as part of the server. In this case, if the function causes a fault, then the database server will be terminated. Because of this, loading and executing functions in an external environment using the LANGUAGE attribute is recommended. If a function causes a fault in an external environment, the database server will continue to run.

EXTERNAL NAME c-call LANGUAGE {C_ESQL32 | C_ESQL64 | C_ODBC32 | C_ODBC64 }
clause    To call a compiled native C function in an external environment instead of within the database server, the stored procedure or function is defined with the EXTERNAL NAME clause followed by the LANGUAGE attribute specifying one of C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64.

When the LANGUAGE attribute is specified, then the library containing the function is loaded by an external process and the external function will execute as part of that external process. In this case, if the function causes a fault, then the database server will continue to run.

CREATE FUNCTION ODBCinsert( IN ProductName CHAR(30),
                                IN ProductDescription CHAR(50))
RETURNS INT
EXTERNAL NAME 'ODBCexternalInsert@extodbc.dll'
LANGUAGE C_ODBC32;

EXTERNAL NAME clr-call LANGUAGE CLR clause    To call a .NET function in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE CLR attribute.

A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a .NET language such as C# or Visual Basic, and the execution of the procedure or function takes place outside the database server (that is, within a separate .NET executable).

CREATE FUNCTION clr_interface( IN p1 INT,
                                 IN p2 UNSIGNED SMALLINT,
                                 IN p3 LONG VARCHAR)
RETURNS INT
EXTERNAL NAME 'CLRlib.dll::CLRproc.Run( int, ushort, string )'
LANGUAGE CLR;

EXTERNAL NAME perl-call LANGUAGE PERL clause    To call a Perl function in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PERL attribute.

A Perl stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Perl and the execution of the procedure or function takes place outside the database server (that is, within a Perl executable instance).

CREATE FUNCTION PerlWriteToConsole( IN str LONG VARCHAR)
RETURNS INT
EXTERNAL NAME '<file=PerlConsoleExample>
EXTERNAL NAME `php-call` LANGUAGE PHP clause  To call a PHP function in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PHP attribute.

A PHP stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in PHP and the execution of the procedure or function takes place outside the database server (that is, within a PHP executable instance).

```
CREATE FUNCTION PHPPopulateTable()
RETURNS INT
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
LANGUAGE PHP;
```

EXTERNAL NAME `java-call` LANGUAGE JAVA clause  To call a Java method in an external environment, the function interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JAVA attribute.

A Java-interfacing stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Java and the execution of the procedure or function takes place outside the database server (that is, within a Java VM).

```
CREATE FUNCTION HelloDemo( IN name LONG VARCHAR )
RETURNS INT
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

The descriptors for arguments and return values from Java methods have the following meanings:

<table>
<thead>
<tr>
<th>Field type</th>
<th>Java data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>byte</td>
</tr>
<tr>
<td>C</td>
<td>char</td>
</tr>
<tr>
<td>D</td>
<td>double</td>
</tr>
<tr>
<td>F</td>
<td>float</td>
</tr>
<tr>
<td>I</td>
<td>int</td>
</tr>
<tr>
<td>J</td>
<td>long</td>
</tr>
<tr>
<td>L class-name;</td>
<td>An instance of the class class-name. The class name must be fully qualified, and any dot in the name must be replaced by a / . For example, java/lang/ String.</td>
</tr>
<tr>
<td>S</td>
<td>short</td>
</tr>
<tr>
<td>V</td>
<td>void</td>
</tr>
<tr>
<td>Field type</td>
<td>Java data type</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Z</td>
<td>Boolean</td>
</tr>
<tr>
<td>[</td>
<td>Use one for each dimension of an array.</td>
</tr>
</tbody>
</table>

**Remarks**

The CREATE FUNCTION statement creates a function in the database. You can create functions for other users by specifying an owner. A function is invoked as part of a SQL expression.

When referencing a temporary table from multiple functions, a potential issue can arise if the temporary table definitions are inconsistent and statements referencing the table are cached.

**Privileges**

To create external functions owned by you, you must have the CREATE PROCEDURE and CREATE EXTERNAL REFERENCE system privileges.

To create external functions owned by others, you must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privileges, and the CREATE EXTERNAL REFERENCE system privilege.

No privilege is required to create temporary functions.

**Side effects**

Automatic commit.

**See also**

- “ALTER FUNCTION statement” on page 439
- “CALL statement” on page 530
- “CREATE FUNCTION statement” on page 594
- “CREATE FUNCTION statement [Web service]” on page 586
- “CREATE PROCEDURE statement [External call]” on page 620
- “DROP FUNCTION statement” on page 758
- “GRANT statement” on page 827
- “SQL Anywhere external environment support” [SQL Anywhere Server - Programming]
- “SQL Anywhere external call interface” [SQL Anywhere Server - Programming]
- “The ESQL and ODBC external environments” [SQL Anywhere Server - Programming]
- “The CLR external environment” [SQL Anywhere Server - Programming]
- “The Java external environment” [SQL Anywhere Server - Programming]
- “The PHP external environment” [SQL Anywhere Server - Programming]
- “The PERL external environment” [SQL Anywhere Server - Programming]
- “SQL data types” on page 89
- “References to temporary tables within procedures” [SQL Anywhere Server - SQL Usage]
Standards and compatibility

- **SQL/2008**  CREATE FUNCTION for an external language environment is a core feature of the SQL/2008 standard, though some of its components supported in SQL Anywhere are optional SQL/2008 language features. A subset of these features include:
  
  ○ The SQL SECURITY clause is optional language feature T324.
  
  ○ The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to a SQL function is language feature T041.
  
  ○ Support for LANGUAGE JAVA is optional SQL/2008 language feature J621.
  
  ○ The ability to create or modify a schema object within an external function, using statements such as CREATE TABLE or DROP TRIGGER, is language feature T653.
  
  ○ The ability to use a dynamic-SQL statement within an external function, including statements such as CONNECT, EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is language feature T654.

  Several clauses of the CREATE FUNCTION statement are vendor extensions. These include:
  
  ○ Support for C_ESQL32, C_ESQL64, C_ODBC32, C_ODBC64, CLR, PERL, and PHP in the LANGUAGES clause are vendor extensions.
  
  ○ The format of external-call is implementation-defined.
  
  ○ The optional DEFAULT clause for a specific routine parameter is a vendor extension.
  
  ○ The optional OR REPLACE clause is a vendor extension.

- **Transact-SQL**  CREATE FUNCTION for an external routine is supported by Adaptive Server Enterprise. Adaptive Server Enterprise only supports LANGUAGE JAVA as the external environment (SQL/2008 language feature J621) for an external function.

CREATE FUNCTION statement [Web service]

Creates a web client function that makes an HTTP or SOAP over HTTP request.

Syntax

```
CREATE [ OR REPLACE ] FUNCTION [ owner.]function-name ( [ parameter, ... ] )
RETURNS data-type
URL url-string
[ HEADER header-string ]
[ SOAPHEADER soap-header-string ]
[ TYPE { 'HTTP' [ GET | POST[:MIME-type] | PUT[:MIME-type] | DELETE | HEAD ] } | 'SOAP' [ :[RPC | DOC] ] } ]
[ NAMESPACE namespace-string ]
[ CERTIFICATE certificate-string ]
[ CLIENTPORT clientport-string ]
[ PROXY proxy-string ]
[ SET protocol-option-string ]
```
**parameter:**

\[
\text{IN parameter-name data-type [ DEFAULT expression ]}
\]

**url-string:**

' { HTTP | HTTPS | HTTPS_FIPS }://[user:password@]hostname[;port][/path]*

**protocol-option-string**

[ http-option-list]

[ , soap-option-list ]

**http-option-list:**

HTTP(

[ CH[UNK]= { ON | OFF | AUTO } ]

[ ; VERSION={1.0 | 1.1};kto=number-of-seconds ]

[ REDIR(COUNT= count, STATUS= status-list) ]
)

**soap-option-list:**

SOAP(OP[ERATION]=soap-operation-name)

**Parameters**

Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type, and must be prefixed by the keyword **IN**, signifying that the argument is an expression that provides a value to the function. However, function parameters are **IN** by default.

When functions are executed, not all parameters need to be specified. If a **DEFAULT** value is provided in the **CREATE FUNCTION** statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, an error is given.

**OR REPLACE clause** Specifying **OR REPLACE** creates a new function, or replaces an existing function with the same name. This clause changes the definition of the function, but preserves existing privileges. You cannot use the **OR REPLACE** clause with temporary functions.

**RETURNS clause** Specify one of the following to define the return type for the SOAP or HTTP function:

- CHAR
- VARCHAR
- LONG VARCHAR
- TEXT
- NCHAR
- NVARCHAR
- LONG NVARCHAR
- NTEXT
- XML
- BINARY
- VARBINARY
- LONG BINARY

The value returned is the body of the HTTP response. No HTTP header information is included. If more information is required, such as status information, use a procedure instead of a function.
The data type does not affect how the HTTP response is processed.

**URL clause**  Use the URL clause only when defining an HTTP or SOAP web services client function. The URL clause specifies the URI of the web service. The optional user name and password parameters provide a means of supplying the credentials needed for HTTP basic authentication. HTTP basic authentication base-64 encodes the user and password information and passes it in the Authentication header of the HTTP request.

Specifying HTTPS_FIPS forces the system to use the FIPS-certified libraries. If HTTPS_FIPS is specified, but no FIPS-certified libraries are present, libraries that are not FIPS-certified are used instead.

For functions of type HTTP:GET, query parameters can be specified within the URL clause in addition to being automatically generated from parameters passed to a procedure.

URL 'http://localhost/service?parm=1

**HEADER clause**  When creating HTTP web service client functions, use this clause to add or modify HTTP request header entries. Only printable ASCII characters can be specified for HTTP headers, and they are case-insensitive.

**SOAPHEADER clause**  When declaring a SOAP web service as a function, use this clause to specify one or more SOAP request header entries. A SOAP header can be declared as a static constant, or can be dynamically set using the parameter substitution mechanism (declaring IN, OUT, or INOUT parameters for hd1, hd2, and so on). A web service function can define one or more IN mode substitution parameters, but cannot define an INOUT or OUT substitution parameter.

**TYPE clause**  The TYPE clause specifies the format used when making the web service request. SOAP:RPC is used when SOAP is specified or no TYPE clause is included. HTTP:POST is used when HTTP is specified.

The TYPE clause allows the specification of a MIME-type for HTTP:GET, HTTP:POST, and HTTP:PUT types. The *MIME-type* specification is used to set the Content-Type request header and set the mode of operation to allow only a single call parameter to populate the body of the request. Only zero or one parameter may remain when making a web service stored function call after parameter substitutions have been processed. Calling a web service function with a NULL value or no parameter (after substitutions) results in a request with no body and a content-length of zero. Parameter names and values (multiple parameters are permitted) are URL encoded within the body of the HTTP request.

Some typical MIME-types include:

- text/plain
- text/html
- text/xml

The keywords for the TYPE clause have the following meanings:

- **HTTP:GET**  By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the URL.

  For example, the following request is produced when a client submits a request from the URL http://localhost/WebServiceName?arg1=param1&arg2=param2:
GET /WebServiceName?arg1=param1&arg2=param2 HTTP/1.1
   // <End of Request - NO BODY>

○ **HTTP:POST** By default, this type uses the application/x-www-form-urlencoded MIME-type for
  encoding parameters specified in the body of a POST request. URL parameters are stored in the body.

  For example, the following request is produced when a client submits a request from the URL
  http://localhost/WebServiceName?arg1=param1&arg2=param2:

  POST /WebServiceName HTTP/1.1
  Content-Type: application/x-www-form-urlencoded
  Content-Length: 19

```
arg1=param1&arg2=param2
   // <End of Request>
```

○ **HTTP:PUT** HTTP:PUT is similar to HTTP:POST, but the HTTP:PUT type does not have a default
  media type.

  The following example demonstrates how to configure a general purpose client procedure that uploads
  data to a SQL Anywhere server running the `put_data.sql` sample:

  ```sql
  ALTER PROCEDURE CPUT("data" LONG VARCHAR, resnm LONG VARCHAR, mediatype
  LONG VARCHAR)
  URL 'http://localhost/resource/!resnm'
  TYPE 'HTTP:PUT:!mediatype';

  CALL CPUT('hello world', 'hello', 'text/plain' );
  ```

○ **HTTP:DELETE** A web service client procedure can be configured to delete a resource located on a
  server. Specifying the media type is optional.

  The following example demonstrates how to configure a general purpose client procedure that deletes
  a resource from a SQL Anywhere server running the `put_data.sql` sample:

  ```sql
  ALTER PROCEDURE CDEL(resnm LONG VARCHAR, mediatype LONG VARCHAR)
  URL 'http://localhost/resource/!resnm'
  TYPE 'HTTP:DELETE:!mediatype';

  CALL CDEL('hello', 'text/plain' );
  ```

○ **HTTP:HEAD** The head method is identical to a GET method but the server does not return a body.
  A media type can be specified.

  ```sql
  ALTER PROCEDURE CHEAD(resnm LONG VARCHAR)
  URL 'http://localhost/resource/!resnm'
  TYPE 'HTTP:HEAD';

  CALL CHEAD( 'hello' );
  ```

○ **SOAP:RPC** This type sets the Content-Type to 'text/xml'. SOAP operations and parameters are
  encapsulated in SOAP envelope XML documents.

○ **SOAP:DOC** This type sets the Content-Type to 'text/xml'. It is similar to the SOAP:RPC type but
  allows you to send richer data types. SOAP operations and parameters are encapsulated in SOAP
  envelope XML documents.
Specifying a MIME-type for the TYPE clause automatically sets the Content-Type header to that MIME-type.

**NAMESPACE clause**  Applies to SOAP client functions only. This clause identifies the method namespace usually required for both SOAP:RPC and SOAP:DOC requests. The SOAP server handling the request uses this namespace to interpret the names of the entities in the SOAP request message body. The namespace can be obtained from the WSDL (Web Services Description Language) of the SOAP service available from the web service server. The default value is the function's URL, up to but not including, the optional path component.

**CERTIFICATE clause**  To make a secure (HTTPS) request, a client must have access to the certificate used by the HTTPS server. The necessary information is specified in a string of semicolon-separated key/value pairs. You can use the file key to specify the file name of the certificate, or you can use the certificate key to specify the server certificate in a string. You cannot specify a file and certificate key together. The following keys are available:

<table>
<thead>
<tr>
<th>Key</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td></td>
<td>The file name of the certificate.</td>
</tr>
<tr>
<td>certificate</td>
<td>cert</td>
<td>The certificate itself.</td>
</tr>
<tr>
<td>certificate_name</td>
<td>cert_name</td>
<td>The name of a certificate stored in the database.</td>
</tr>
<tr>
<td>company</td>
<td>co</td>
<td>The company specified in the certificate.</td>
</tr>
<tr>
<td>unit</td>
<td></td>
<td>The company unit specified in the certificate.</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>The common name specified in the certificate.</td>
</tr>
</tbody>
</table>

Certificates are required only for requests that are directed to an HTTPS server, or for requests that can be redirected from a non-secure to a secure server. Only PEM formatted certificates are supported.

**CLIENTPORT clause**  Identifies the port number on which the HTTP client function communicates using TCP/IP. It is provided for and recommended only for connections across firewalls, as firewalls filter according to the TCP/UDP port. You can specify a single port number, ranges of port numbers, or a combination of both; for example, `CLIENTPORT '85,90-97'`.

**PROXY clause**  Specifies the URI of a proxy server. For use when the client must access the network through a proxy. This clause indicates that the function is to connect to the proxy server and send the request to the web service through it.

**SET clause**  Specifies protocol-specific behavior options for HTTP and SOAP. The following list describes the supported SET options. CHUNK and VERSION apply to the HTTP protocol, and OPERATION applies to the SOAP protocol. Parameter substitution is supported for this clause.
HTTP(CHUNK=option)  (HTTP or SOAP) This option allows you to specify whether to use chunking. Chunking allows HTTP messages to be broken up into several parts. Possible values are ON (always chunk), OFF (never chunk), and AUTO (chunk only if the contents, excluding auto-generated markup, exceeds 8196 bytes). For example, the following SET clause enables chunking:

```
SET 'HTTP(CHUNK=ON)'
```

If the CHUNK option is not specified, the default behavior is AUTO. If a chunked request fails in AUTO mode with a status of 505 HTTP Version Not Supported, or with 501 Not Implemented, or with 411 Length Required, the client retries the request without chunked transfer-coding.

Set the CHUNK option to OFF (never chunk) if the HTTP server does not support chunked transfer-coded requests.

Since CHUNK mode is a transfer encoding supported starting in HTTP version 1.1, setting CHUNK to ON requires that the version (VER) be set to 1.1, or not be set at all, in which case 1.1 is used as the default version.

HTTP(VERSION=ver;kto=number-of-seconds)  (HTTP or SOAP) This option allows you to specify the version of HTTP protocol that is used for the format of the HTTP message. For example, the following SET clause sets the HTTP version to 1.1:

```
SET 'HTTP(VERSION=1.1)'
```

Possible values are 1.0 and 1.1. If VERSION is not specified:

- if CHUNK is set to ON, 1.1 is used as the HTTP version
- if CHUNK is set to OFF, 1.0 is used as the HTTP version
- if CHUNK is set to AUTO, either 1.0 or 1.1 is used, depending on whether the client is sending in CHUNK mode

kto=number-of-seconds  Specifies the keep-alive timeout criteria (kto), permitting a web client procedure to instantiate and cache a keep-alive HTTP/HTTPS connection for a period of time. To cache an HTTP keep-alive connection, the HTTP version must be set to 1.1 and kto set to a non-zero value. kto may be useful, for HTTPS connections particularly, if you notice a significant performance difference between HTTP and HTTPS connections. A database connection can only cache a single keep-alive HTTP connection. Subsequent calls to a web client procedure using the same URI reuse the keep-alive connection. Therefore, the executing web client call must have a URI whose scheme, destination host and port match that of the cached URI, and the HEADER clause must not specify Connection: close. When kto is not specified, or is set to zero, HTTP/HTTPS connections are not cached.

REDIR(COUNT=count, STATUS=status-list)  (HTTP or SOAP) The HTTP response status codes such as 302 Found and 303 See Other are used to redirect web applications to a new URI, particularly after an HTTP POST has been performed. For example, a client request could be:

```
GET /people/alice HTTP/1.1
Host: www.example.com
```
The web server response could be:

```
HTTP/1.1 302 Found
Location: http://www.example.com/people/alice.en.html
```

In response, the client would send another HTTP request to the new URI. The REDIR option allows you to control the maximum number of redirections allowed and which HTTP response status codes to automatically redirect.

For example, `SET 'REDIR(count=3, status=301,307)'` allows a maximum limit of 3 redirections and permits redirection for 301 and 307 statuses. If one of the other redirection status codes such as 302 or 303 is received, an error is issued (SQLCODE -983).

The default redirection limit `count` is 5. By default, an HTTP client procedure will automatically redirect in response to all HTTP redirection status codes (301, 302, 303, 307). To disallow all redirection status codes, use `SET REDIR(COUNT=0)`. In this mode, a redirection response does not result in an error (SQLCODE -983). Instead, a result set is returned with the HTTP status and response headers. This permits a caller to conditionally reissue the request based on the URI contained in the `Location` header.

A web service procedure specifying a POST HTTP method which receives a 303 See Other status issues a redirect request using the GET HTTP method.

The `Location` header can contain either an absolute path or a relative path. The HTTP client procedure handles either. The header can also include query parameters and these are forwarded to the redirected location. For example, if the header contained parameters such as the following, the subsequent GET or a POST includes these parameters.

```
Location: alternate_service?a=1&b=2
```

In the above example, the query parameters are `a=1&b=2`.

- `SOAP(OP[ERATION]=soap-operation-name)` (SOAP only) This option allows you to specify the name of the SOAP operation, if it is different from the name of the procedure you are creating. The value of `OPERATION` is analogous to the name of a remote procedure call. For example, if you wanted to create a procedure called accounts_login that calls a SOAP operation called login, you would specify something like the following:

```sql
CREATE FUNCTION accounts_login(
    name LONG VARCHAR,
    pwd LONG VARCHAR);
SET 'SOAP(OPERATION=login)';
```

If the `OPERATION` option is not specified, the name of the SOAP operation must match the name of the procedure you are creating.

The following statement shows how several `protocol-option-string` settings are combined in the same `SET` clause:

```sql
CREATE FUNCTION accounts_login(
    name LONG VARCHAR,
```
pwd LONG VARCHAR );
SET 'HTTP ( CHUNK=ON; VERSION=1.1 ), SOAP( OPERATION=login )'
...

Remarks
The CREATE FUNCTION statement creates a web services function in the database. A function can be created for another user by specifying an owner name.

Parameter values are passed as part of the request. The syntax used depends on the type of request. For HTTP:GET, the parameters are passed as part of the URL; for HTTP:POST requests, the values are placed in the body of the request. Parameters to SOAP requests are always bundled in the request body.

Privileges
You must have the CREATE PROCEDURE system privilege to create functions owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create functions owned by others.

You must have the CREATE EXTERNAL REFERENCE system privilege to create an external function.

No privilege is required to create temporary functions.

Side effects
Automatic commit.

See also
- “SQL data types” on page 89
- “Tutorial: Using SQL Anywhere to access a SOAP/DISH service” [SQL Anywhere Server - Programming]
- “HTTP and SOAP request structures” [SQL Anywhere Server - Programming]
- “HTTP request header management” [SQL Anywhere Server - Programming]
- “ALTER FUNCTION statement” on page 439
- “CREATE FUNCTION statement” on page 594
- “CREATE FUNCTION statement [External call]” on page 580
- “CREATE PROCEDURE statement” on page 639
- “CREATE PROCEDURE statement [Web service]” on page 628
- “DROP FUNCTION statement” on page 758
- “RETURN statement” on page 940
- “Variables supplied to web services” [SQL Anywhere Server - Programming]
- “Tutorial: Create a web server and access it from a web client” [SQL Anywhere Server - Programming]
- “SQL Anywhere as an HTTP web server” [SQL Anywhere Server - Programming]
- “Web client application development” [SQL Anywhere Server - Programming]
- “remote_idle_timeout option” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.
Examples

The following statement creates a function named cli_test1 that returns images from the get_picture service running on localhost:

```sql
CREATE FUNCTION cli_test1( image LONG VARCHAR )
RETURNS LONG BINARY
URL 'http://localhost/get_picture'
TYPE 'HTTP:GET';
```

The following statement issues an HTTP request with the URL http://localhost/get_picture?
image=widget:

```sql
SELECT cli_test1( 'widget' );
```

The following statement uses a substitution parameter to allow the request URL to be passed as an input parameter. The secure HTTPS request uses a certificate stored in the database. The SET clause is used to turn off CHUNK mode transfer-encoding.

```sql
CREATE CERTIFICATE client_cert
FROM FILE 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\Certificates\rsaroot.crt';

CREATE FUNCTION cli_test2( image LONG VARCHAR, myurl LONG VARCHAR )
RETURNS LONG BINARY
URL '!myurl'
CERTIFICATE 'certificate_name=client_cert'
TYPE 'HTTPS:GET'
SET 'HTTP(CH=OFF)'
HEADER 'ASA-ID';
```

The following statement issues an HTTP request with the URL http://localhost/get_picture?
image=widget:

```sql
CREATE VARIABLE a_binary LONG BINARY;
SET a_binary = cli_test2( 'widget', 'https://localhost/get_picture' );
SELECT a_binary;
```

CREATE FUNCTION statement

Creates a user-defined SQL function in the database.

Syntax

```
CREATE [ OR REPLACE ] [ TEMPORARY ] FUNCTION [ owner.]function-name
( [ parameter, ... ]
[ SQL SECURITY { INVOKER | DEFINER } ]
RETURNS data-type
[ ON EXCEPTION RESUME ]
[ [ NOT ] DETERMINISTIC ]
compound-statement | AS tsqI-compound-statement | AT location-string

parameter :
  [ IN ] parameter-name data-type [ DEFAULT expression ]
```

- **Transact-SQL**  Not supported by Adaptive Server Enterprise.
Parameters

Parameter names must conform to the rules for database identifiers. They must have a valid SQL data type, and must be prefixed by the keyword IN, signifying that the argument is an expression that provides a value to the function. However, function parameters are IN by default.

When functions are executed, not all parameters need to be specified. If a DEFAULT value is provided in the CREATE FUNCTION statement, missing parameters are assigned the default values. If an argument is not provided by the caller and no default is set, an error is given.

**OR REPLACE clause** Specifying CREATE OR REPLACE FUNCTION creates a new function, or replaces an existing function with the same name. When a function is replaced, the definition of the function is changed but the existing privileges are preserved.

You cannot use the OR REPLACE clause with temporary functions.

**TEMPORARY keyword** Specifying CREATE TEMPORARY FUNCTION means that the function is visible only by the connection that created it, and that it is automatically dropped when the connection is dropped. Temporary functions can also be explicitly dropped. You cannot perform ALTER, GRANT, or REVOKE on them, and, unlike other functions, temporary functions are not recorded in the catalog or transaction log.

Temporary functions execute with the privileges of their creator (current user) or specified owner. You can specify an owner for a temporary function when:

- the temporary function is created within a permanent stored procedure
- the owner of the temporary function and permanent stored procedure is the same

To drop the owner of a temporary function, you must drop the temporary function first.

Temporary functions can be created and dropped when connected to a read-only database.

You cannot use the OR REPLACE clause with temporary functions.

**RETURNS clause** Use the RETURNS clause to specify the data type for the result of the function.

**SQL SECURITY clause** The SQL SECURITY clause defines whether the function is executed as the INVOKER (the user who is calling the function), or as the DEFINER (the user who owns the function). The default is DEFINER.

**ON EXCEPTION RESUME clause** Use Transact-SQL-like error handling.

**[ NOT ] DETERMINISTIC clause** Use this clause to indicate whether functions are deterministic or non-deterministic. If this clause is omitted, then the deterministic behavior of the function is unspecified (the default).
If a function is declared as DETERMINISTIC, it should return the same value every time it is invoked with the same set of parameters.

If a function is declared as NOT DETERMINISTIC, then it is not guaranteed to return the same value for the same set of parameters. A function declared as NOT DETERMINISTIC is re-evaluated each time it is called in a query. This clause must be used when it is known that the function result for a given set of parameters can vary.

Also, functions that have side effects such as modifying the underlying data should be declared as NOT DETERMINISTIC. For example, here is an example of a function that generates primary key values and is used in an INSERT...SELECT statement; it is declared NOT DETERMINISTIC. The tables that are referenced are fictitious.

```sql
CREATE FUNCTION keygen( increment INTEGER )
RETURNS INTEGER
NOT DETERMINISTIC BEGIN
  DECLARE keyval INTEGER;
  UPDATE counter SET x = x + increment;
  SELECT counter.x INTO keyval FROM counter;
  RETURN keyval END
INSERT INTO new_table
SELECT keygen(1), ...
FROM old_table;
```

Functions can be declared as DETERMINISTIC if they always return the same value for given input parameters.

**compound-statement**  A set of SQL statements bracketed by BEGIN and END, and separated by semicolons.

**AS clause**  `tsql-compound-statement` is a batch of Transact-SQL statements.

**AT clause**  Create a proxy function on the current database for a remote function specified by `location-string`. The AT clause supports the semicolon (;) as a field delimiter in `location-string`. If no semicolon is present, a period is the field delimiter. The use of semicolons allows file names and extensions to be used in the database and owner fields.

The string in the AT clause can also contain local or global variable names enclosed in braces (`{variable-name}`). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains `'bostonase.master.dbo.{@myfunction}'` indicates that `@myfunction` is a SQL variable and that the current contents of the `@myfunction` variable should be substituted when the remote procedure is used.

A proxy function can return any data type except DECIMAL, NUMERIC, LONG VARCHAR, LONG NVARCHAR, LONG BINARY, XML, or any spatial data type.

**Remarks**

The CREATE FUNCTION statement creates a function in the database. A function can be created for another user by specifying an owner name. Subject to privileges, a function can be used in exactly the same way as other non-aggregate functions.
When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker. Therefore, make sure to qualify all object names (tables, procedures, and so on) with their appropriate owner.

All functions are treated as deterministic unless they are declared NOT DETERMINISTIC. Deterministic functions return a consistent result for the same parameters, and are free of side effects. That is, the database server assumes that two successive calls to the same function with the same parameters returns the same result, and does not have any unwanted side effects on the query's semantics.

If a function returns a result set, it cannot also set output parameters or return a return value.

**Privileges**

You must have the CREATE PROCEDURE system privilege to create functions owned by you.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create functions owned by others.

You must also have the CREATE EXTERNAL REFERENCE system privilege to create an external function.

No privilege is required to create temporary functions.

**Side effects**

Automatic commit, even for temporary functions.

**See also**

- “ALTER FUNCTION statement” on page 439
- “CREATE FUNCTION statement [External call]” on page 580
- “CREATE FUNCTION statement [Web service]” on page 586
- “BEGIN statement” on page 523
- “CREATE PROCEDURE statement” on page 639
- “DROP FUNCTION statement” on page 758
- “RETURN statement” on page 940
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]
- “Creating a user-defined function” [SQL Anywhere Server - SQL Usage]
- “SQL data types” on page 89
- “Transact-SQL batches” [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**

- **SQL/2008**  CREATE FUNCTION is a core feature of the SQL/2008 standard, though some of its components supported in SQL Anywhere are optional SQL language features. A subset of these features includes:
  - The SQL SECURITY clause is optional language feature T324.
  - The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to a SQL function is language feature T041.
The ability to create or modify a schema object within a SQL function, using statements such as CREATE TABLE or DROP TRIGGER, is language feature T651.

The ability to use a dynamic-SQL statement within a SQL function, including statements such as EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is language feature T652.

Several clauses of the CREATE FUNCTION statement are vendor extensions. These include:

- The TEMPORARY clause.
- The ON EXCEPTION RESUME clause.
- The optional DEFAULT clause for a specific routine parameter.
- The specification of a Transact-SQL function using the AS clause.
- The optional OR REPLACE clause.

Transact-SQL CREATE FUNCTION is supported by Adaptive Server Enterprise. Adaptive Server Enterprise does not support the optional IN keyword for function parameters.

Examples

The following function concatenates a firstname string and a lastname string.

```sql
CREATE FUNCTION fullname(
    firstname CHAR(30),
    lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
    DECLARE name CHAR(61);
    SET name = firstname || ' ' || lastname;
    RETURN (name);
END;
```

The following example replaces the fullname function created in the first example. After replacing the function, the local variable name is removed:

```sql
CREATE OR REPLACE FUNCTION fullname(
    firstname CHAR(30),
    lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
    RETURN ( firstname || ' ' || lastname );
END;
```

The following examples illustrate the use of the fullname function.

Return a full name from two supplied strings:

```sql
SELECT fullname ( 'joe', 'smith' );
```

<table>
<thead>
<tr>
<th>fullname('joe', 'smith')</th>
<th>joe smith</th>
</tr>
</thead>
</table>
List the names of all employees:

```
SELECT fullname ( GivenName, Surname )
FROM GROUPO.Employees;
```

<table>
<thead>
<tr>
<th>fullname (GivenName, Surname)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fran Whitney</td>
</tr>
<tr>
<td>Matthew Cobb</td>
</tr>
<tr>
<td>Philip Chin</td>
</tr>
<tr>
<td>Julic Jordan</td>
</tr>
</tbody>
</table>

The following function uses Transact-SQL syntax:

```
CREATE FUNCTION DoubleIt( @Input INT )
RETURNS INT
AS
BEGIN
    DECLARE @Result INT
    SELECT @Result = @Input * 2
    RETURN @Result
END;
```

The statement `SELECT DoubleIt( 5 )` returns a value of 10.

**CREATE INDEX statement**

Creates an index on a specified table or materialized view.

**Syntax 1 - Creating an index on a table**

```
CREATE [ VIRTUAL ] [ UNIQUE ] [ CLUSTERED ] INDEX [ IF NOT EXISTS ] index-name
ON [ owner. ] table-name
    ( column-name [ ASC | DESC ], ...
    | function-name ( argument, [ ... ] ) AS column-name )
    [ WITH NULLS [ NOT ] DISTINCT ]
    { IN | ON } dbspace-name
    [ FOR OLAP WORKLOAD ]
```

**Syntax 2 - Creating an index on a materialized view**

```
CREATE [ VIRTUAL ] [ UNIQUE ] [ CLUSTERED ] INDEX [ IF NOT EXISTS ] index-name
ON [ owner. ] materialized-view-name
    ( column-name [ ASC | DEC SC ], ...)
    [ WITH NULLS NOT DISTINCT ]
    { IN | ON } dbspace-name
    [ FOR OLAP WORKLOAD ]
```
Parameters

**VIRTUAL clause** The VIRTUAL keyword is primarily for use by the Index Consultant. A virtual index mimics the properties of a real physical index during the evaluation of execution plans by the Index Consultant and when the PLAN function is used. You can use virtual indexes together with the PLAN function to explore the performance impact of an index, without the often time-consuming and resource-consuming effects of creating a real index.

Virtual indexes are not visible to other connections, and are dropped when the connection is closed. Virtual indexes are not used when evaluating plans for the actual execution of queries, and so do not interfere with performance.

Virtual indexes have a limit of four columns.

**UNIQUE clause** The UNIQUE attribute ensures that there will not be two rows in the table or materialized view with identical values in all the columns in the index. If you specify UNIQUE, but do not specify WITH NULLS NOT DISTINCT, each index key must be unique or contain a NULL in at least one column. For example, two entries ('a', NULL) and ('a', NULL) are each considered unique.

If you specify UNIQUE...WITH NULLS NOT DISTINCT, then the index key must be unique regardless of the NULL values. For example, two entries ('a', NULL) and ('a', NULL) are considered equal, not unique.

There is a difference between a unique constraint and a unique index. Columns of a unique index are allowed to be NULL, while columns in a unique constraint are not. A foreign key can reference either a primary key or a unique constraint, but not a unique index, because it can include multiple instances of NULL.

It is recommended that you do not use approximate data types such as FLOAT and DOUBLE for primary keys or for columns in unique constraints. Approximate numeric data types are subject to rounding errors after arithmetic operations.

Spatial columns cannot be included in a unique index.

**CLUSTERED clause** The CLUSTERED attribute causes rows to be stored in an approximate key order corresponding to the index. While the database server makes an attempt to preserve key order, total clustering is not guaranteed.

If a clustered index exists, the LOAD TABLE statement inserts rows in the order of the index key, and the INSERT statement attempts to put new rows on the same page as the one containing adjacent rows, as defined by the key order.

**IF NOT EXISTS clause** When the IF NOT EXISTS attribute is specified and the named index already exists, no changes are made and an error is not returned.

**ASC | DESC clause** Columns are sorted in ascending (increasing) order unless descending (DESC) is explicitly specified. An index is used for both an ascending and a descending ORDER BY, whether the index was ascending or descending. However, if an ORDER BY is performed with mixed ascending and descending attributes, an index is used only if the index was created with the same ascending and descending attributes.
**function-name**  The function-name clause creates an index on a function. This clause cannot be used on declared temporary tables or materialized views.

This form of the CREATE INDEX statement is a convenience method that carries out the following operations:

1. Adds a computed column named `column-name` to the table. The column is defined with a COMPUTE clause that is the specified function, along with any specified arguments. See the COMPUTE clause of the CREATE TABLE statement for restrictions on the type of function that can be specified. The data type of the column is based on the result type of the function.

2. Populates the computed column for the existing rows in the table.

3. Creates an index on the column.

   Dropping the index does not cause the associated computed column to be dropped.

**IN | ON clause**  By default, the index is placed in the same database file as its table or materialized view. You can place the index in a separate database file by specifying a dbspace name in which to put the index. This feature is useful mainly for large databases to circumvent file size limitations, or for performance improvements that might be achieved by using multiple disk devices.

If the new index can share the physical index with an existing logical index, the IN clause is ignored.

**WITH NULLS NOT DISTINCT clause**  This clause can only be specified if you are declaring the index to be UNIQUE and allows you to specify that NULLs in index keys are not unique. For more information, see the UNIQUE clause.

**FOR OLAP WORKLOAD clause**  When you specify FOR OLAP WORKLOAD, the database server performs certain optimizations and gathers statistics on the key to help improve performance for OLAP workloads. Performance improvements are most noticeable when the optimization_workload is set to OLAP.

Remarks

Syntax 1 is for use with tables; Syntax 2 is for use with materialized views.

Indexes can improve database performance. SQL Anywhere uses physical and logical indexes. A physical index is the actual indexing structure as it is stored on disk. A logical index is a reference to a physical index. If you create an index that is identical in its physical attributes to an existing index, the database server creates a logical index that shares the existing physical index. In general, indexes created by users are considered logical indexes. The database server creates physical indexes as required to implement logical indexes, and can share the same physical index among several logical indexes.

The CREATE INDEX statement creates a sorted index on the specified columns of the named table or materialized view. Indexes are automatically used to improve the performance of queries issued to the database, and to sort queries with an ORDER BY clause. Once an index is created, it is never referenced in a SQL statement again except to validate it (VALIDATE INDEX), alter it (ALTER INDEX), delete it (DROP INDEX), or in a hint to the optimizer.
Index ownership
There is no way of specifying the index owner in the CREATE INDEX
statement. Indexes are always owned by the owner of the table or materialized view.

Indexes on views
You can create indexes on materialized views, but not on regular views.

Index name space
The name of each index must be unique for a given table or materialized view.

Exclusive use
CREATE INDEX is prevented whenever the statement affects a table or
materialized view currently being used by another connection. CREATE INDEX can be time
consuming and the database server will not process requests referencing the same table while the
statement is being processed.

Automatically created indexes
SQL Anywhere automatically creates indexes for primary key,
foreign key, and unique constraints. These automatically created indexes are held in the same database
file as the table.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use
either statement or transaction snapshots.

Privileges
To create an index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

To create an index on a materialized view, you must be the owner of the materialized view, or have one of
the following privileges:

- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

Side effects
Automatic commit in most cases. If the auto_commit_on_create_local_temp_index option is set to Off,
there is no commit before creating an index on a local temporary table. Creating an index on a function
(an implicit computed column) causes a checkpoint.

Column statistics are updated (or created if they do not exist).
See also

- “Creating an index” [SQL Anywhere Server - SQL Usage]
- “auto_commit_on_create_local_temp_index option” [SQL Anywhere Server - Database Administration]
- “DROP INDEX statement” on page 759
- “CREATE STATISTICS statement” on page 682
- “optimization_workload option” [SQL Anywhere Server - Database Administration]
- “Advanced: Logical and physical indexes” [SQL Anywhere Server - SQL Usage]
- “Indexes” [SQL Anywhere Server - SQL Usage]
- “Obtaining Index Consultant recommendations for a query” [SQL Anywhere Server - SQL Usage]
- “Index Consultant” [SQL Anywhere Server - SQL Usage]
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]
- “Clustered indexes” [SQL Anywhere Server - SQL Usage]
- “Computed columns” [SQL Anywhere Server - SQL Usage]
- “OLAP support” [SQL Anywhere Server - Database Administration]
- “SQL Anywhere size and number limitations” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

Create a two-column index on the Employees table.

```
CREATE INDEX employee_name_index
ON GROUPO.Employees
( Surname, GivenName );
```

Create an index on the SalesOrderItems table for the ProductID column.

```
CREATE INDEX item_prod
ON GROUPO.SalesOrderItems
( ProductID );
```

Use the SORTKEY function to create an index on the Description column of the Products table, sorted according to a Russian collation. As a side effect, the statement adds a computed column desc_ru to the table. For this example to succeed, you must also have SELECT privilege on the Products table.

```
CREATE INDEX ix_desc_ru
ON GROUPO.Products (
    SORTKEY( Description, 'rusdict' )
    AS desc_ru );
```

CREATE LDAP SERVER statement

Creates an LDAP server configuration object.

Syntax

```
CREATE LDAP SERVER ldapua-server-name
[ ldapua-server-attrs ... ]
[ WITH ACTIVATE ]
```
ldapua-server-attrs :
SEARCH DN search-dn-attributes ...
| AUTHENTICATION URL { url-string | NULL }
| CONNECTION TIMEOUT timeout-value
| CONNECTION RETRIES retry-value
| TLS { ON | OFF }

search-dn-attributes :
| URL { url-string | NULL }
| ACCESS ACCOUNT { dn-string | NULL }
| IDENTIFIED BY { password | NULL }
| IDENTIFIED BY ENCRYPTED { encrypted-password | NULL }

Parameters

SEARCH DN clause  There is no default value for any parameter in the SEARCH DN clause.

○ **URL**  Use this clause to specify to identify the host (by name or by IP address), port number, and search to be performed to do the lookup of the LDAP Distinguished Name (DN) for a given user ID.  *url-string* is validated for correct LDAP URL syntax before it is stored in ISYSLDAPSERVER. The maximum size for this string is 1024 bytes.


○ **ACCESS ACCOUNT**  Use this clause to specify the DN used by the database server to connect to the LDAP server. This is not a SQL Anywhere user, but a user created in the LDAP server specifically for logging in to the LDAP server. This user must have permissions within the LDAP server to search for DNs by user ID in the locations specified in the SEARCH DN URL clause. The maximum size for this string is 1024 bytes.

○ **IDENTIFIED BY**  Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT. The maximum size is 255 bytes and cannot be set to NULL.

○ **IDENTIFIED BY ENCRYPTED**  Use this clause to specify the password associated with the user identified by ACCESS ACCOUNT, provided in encrypted form, and is a binary value stored somewhere on disk. The maximum size of the binary is 289 bytes, and cannot be set to NULL. IDENTIFIED BY ENCRYPTED allows the password to be retrieved and used, without it becoming known.

AUTHENTICATION URL clause  Use this clause to specify the *url-string* that identifies the host by name or IP address, and the port number of the LDAP server to use to authenticate a user. The DN of the user obtained from a prior DN search and the user password are used to bind a new connection to the authentication URL. A successful connection to the LDAP server is considered proof of the identity of the connecting user. There is no default value for this parameter. For size limits to this string, see the SYSLDAPSERVER system view.

CONNECTION TIMEOUT clause  Use this clause to specify the connection timeout, in milliseconds, to the LDAP server, both for searches for the DN and for authentication. The default value is 10 seconds.

CONNECTION RETRIES clause  Use this clause to specify the number of retries for connections to the LDAP server, both for searches for the DN and for authentication. The valid range of values is 1-60. The default is 3.
**TLS clause**  Use this clause to specify the use of the TLS protocol on connections to the LDAP server, both for the DN searches and for authentication. The valid values are ON or OFF. The default is OFF. Use the Secure LDAP protocol by using `ldaps://` to begin the URL instead of `ldap://`. The TLS option must be set to OFF when using Secure LDAP.

**WITH ACTIVATE clause**  Use this clause to activate the LDAP server for immediate use. This clause permits the definition and activation of LDAP User Authentication in one statement, changing the state of the new LDAP server to READY.

**Remarks**
None

**Privileges**
You must have the MANAGE ANY LDAP SERVER system privilege.

**Side effects**
Automatic commit.

**See also**
- “LDAP user authentication” [SQL Anywhere Server - SQL Usage]
- “ALTER LDAP SERVER statement” on page 442
- “DROP LDAP SERVER statement” on page 760
- “VALIDATE LDAP SERVER statement” on page 1044

**Standards and compatibility**
- SQL/2008  Vendor extension.

**Example**
This example sets search parameters, authentication URL, 3 second timeout, and activates the LDAP server so it can begin authenticating users. A connection is made to the LDAP server without TLS or SECURE LDAP protocols. In addition to the privileges required to execute the CREATE LDAP SERVER statement, you must also have the SET ANY SECURITY system privilege to set the login_mode option in the following example.

```sql
SET OPTION PUBLIC.login_mode = 'Standard,LDAPUA';
CREATE LDAP SERVER apps_primary
SEARCH DN
  URL 'ldap://voyager:389/dc=MyCompany,dc=com??sub?cn=*'
  ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
  IDENTIFIED BY 'Secret99Password'
  AUTHENTICATION URL 'ldap://voyager:389/'
  CONNECTION TIMEOUT 3000
WITH ACTIVATE;
```

This example uses the same search parameters, but specifies `ldaps://` so that a Secure LDAP connection is established with the LDAP server on host voyager, port 636. Only LDAP clients using the Secure LDAP protocol may connect on this port. The database security option Trusted_certificate_file must be set with a filename containing the certificate of the Certificate Authority (CA) that signed the certificate used by the LDAP server at `ldaps://voyager:636`. During the handshake with the...
LDAP server, the certificate presented by the LDAP server is verified by the database server to ensure that it is signed by one of the certificates listed in the file. The ACCESS ACCOUNT and IDENTIFIED BY parameters provided to the LDAP server are verified by the LDAP server as well.

```
SET OPTION PUBLIC.login_mode = 'Standard,LDAPUA';
SET OPTION PUBLIC.trusted_certificates_file = '/opt/sybase/shared/trusted.txt';
CREATE LDAP SERVER secure_primary
  SEARCH DN
    URL 'ldaps://voyager:636/dc=MyCompany,dc=com??sub?cn=*
    ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
    IDENTIFIED BY 'Secret99Password'
    AUTHENTICATION URL 'ldaps://voyager:636/'
    CONNECTION TIMEOUT 3000
    WITH ACTIVATE;
```

CREATE LOCAL TEMPORARY TABLE statement

Creates a local temporary table within a procedure that persists after the procedure completes and until it is either explicitly dropped, or until the connection terminates.

**Syntax**

```
CREATE LOCAL TEMPORARY TABLE IF NOT EXISTS table-name
  ( { column-definition [ column-constraint ... ] | table-constraint | pctfree }, ... )
  [ ON COMMIT { DELETE | PRESERVE } ROWS | NOT TRANSACTIONAL ]

  pctfree : PCTFREE percent-free-space

  percent-free-space : integer
```

**Parameters**

- **IF NOT EXISTS clause**  
  No changes are made if the named table already exists, and an error is not returned.

- **ON COMMIT clause**  
  By default, the rows of a temporary table are deleted on a COMMIT. You can use the ON COMMIT clause to preserve rows on a COMMIT.

- **NOT TRANSACTIONAL clause**  
  The NOT TRANSACTIONAL clause provides performance improvements in some circumstances because operations on non-transactional temporary tables do not cause entries to be made in the rollback log. For example, NOT TRANSACTIONAL may be useful if procedures that use the temporary table are called repeatedly with no intervening COMMITs or ROLLBACKs.

**Remarks**

In a procedure, use the CREATE LOCAL TEMPORARY TABLE statement, instead of the DECLARE LOCAL TEMPORARY TABLE statement, when you want to create a table that persists after the procedure completes. Local temporary tables created using the CREATE LOCAL TEMPORARY TABLE statement remain until they are either explicitly dropped, or until the connection closes.

Tables created using CREATE LOCAL TEMPORARY TABLE do not appear in the SYSTABLE view of the system catalog.
Local temporary tables created in IF statements using CREATE LOCAL TEMPORARY TABLE also persist after the IF statement completes.

**Privileges**

None.

**Side effects**

None.

**See also**

- “CREATE TABLE statement” on page 690
- “DECLARE LOCAL TEMPORARY TABLE statement” on page 735
- “Compound statements” [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**

- **SQL/2008** CREATE LOCAL TEMPORARY TABLE is part of optional language feature F531 of the SQL/2008 standard. The PCTFREE and NOT TRANSACTIONAL clauses are vendor extensions. The column and constraint definitions defined by the statement may also include vendor extension syntax. In SQL/2008, the standard stipulates that tables created via the CREATE LOCAL TEMPORARY TABLE statement appear in the system catalog; this is not the case with SQL Anywhere.

- **Transact-SQL** CREATE LOCAL TEMPORARY TABLE is not supported by Adaptive Server Enterprise. In Sybase Adaptive Server Enterprise, one creates a temporary table using the CREATE TABLE statement with a table name that begins with the special character #.

**Example**

The following example creates a local temporary table called TempTab:

```
CREATE LOCAL TEMPORARY TABLE TempTab ( number INT )
ON COMMIT PRESERVE ROWS;
```

**CREATE LOGIN POLICY statement**

Creates a login policy.

**Syntax**

```
CREATE LOGIN POLICY policy-name policy-options

policy options :
policy-option [ policy-option ... ]

policy-option :
policy-option-name = policy-option-value
```

```
<table>
<thead>
<tr>
<th>policy-option-value :</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ UNLIMITED</td>
</tr>
</tbody>
</table>
```
**Parameters**

- **policy-name**  The name of the login policy.

- **policy-option-name**  The name of the login policy option.

- **policy-option-value**  The value assigned to the login policy option. If you specify UNLIMITED, no limits are imposed.

<table>
<thead>
<tr>
<th>Policy-option-name</th>
<th>Description</th>
<th>Default value</th>
<th>Applies to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_unlock_time</td>
<td>The time period after which locked accounts are automatically unlocked.</td>
<td>Unlimi-ted</td>
<td>All users except those with the MANAGE ANY USER system privilege</td>
</tr>
<tr>
<td>change_password_dual_control</td>
<td>When the value for this option is ON, setting the password requires two administrators.</td>
<td>OFF</td>
<td>All users</td>
</tr>
<tr>
<td></td>
<td>The setting for the verify_password_function option is ignored if this option is set to ON because the password is configured separately in two parts.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No verification is performed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ldap_primary_server</td>
<td>The name of the primary LDAP server.</td>
<td>(none)</td>
<td>All users</td>
</tr>
<tr>
<td>ldap_secondary_server</td>
<td>The name of the secondary LDAP server.</td>
<td>(none)</td>
<td>All users</td>
</tr>
<tr>
<td>ldap_auto_failback_period</td>
<td>The time period in minutes after which automatic failback to primary server is attempted.</td>
<td>15 minutes</td>
<td>All users</td>
</tr>
<tr>
<td>ldap_failover_to_std</td>
<td>Whether to permit authentication with Standard authentication when authentication with the LDAP server fails due to failure to locate the Distinguished Name (DN) for a user, lack of system resources, network outage, connection timeouts, or similar system failures. This setting does not permit an actual authentication failure returned from an LDAP server to fail over to Standard authentication (as is the case when the user is located but the supplied password does not match).</td>
<td>ON</td>
<td>All users</td>
</tr>
<tr>
<td>Policy-option-name</td>
<td>Description</td>
<td>Default value</td>
<td>Applies to:</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>ldap_refresh_dn</td>
<td>At the time this policy option is specified by a CREATE LOGIN POLICY or ALTER LOGIN POLICY statement, the current time value is stored with the login policy. This value is the timestamp against which user authentication compares the user_dn_cached_at value found for the user in ISYSUSER. If the value in the policy is newer than the user_dn_cached_at value in ISYSUSER, a search for a user's Distinguished Name (DN) is done to refresh the user_dn value in ISYSUSER. The value NOW is the only valid value to assign to this policy option. All others result in an error. The value is in Coordinated Universal Time (UTC) and is stored as a string in the server default format.</td>
<td>(none)</td>
<td>All users</td>
</tr>
<tr>
<td>locked</td>
<td>If the value for this option is ON, users are not allowed to establish new connections. The reason_locked column of the sa_get_user_status system procedure returns a string generated by the database server that shows why a user is locked.</td>
<td>OFF</td>
<td>All users except those with the MANAGE ANY USER system privilege</td>
</tr>
<tr>
<td>max_connections</td>
<td>The maximum number of concurrent connections allowed for a user.</td>
<td>Unlimited</td>
<td>All users except those with the SERVER OPERATOR or DROP CONNECTION system privilege</td>
</tr>
<tr>
<td>max_failed_login_attempts</td>
<td>The maximum number of failed attempts since the last successful attempt to log in before the user is locked. Users with SYS_AUTH_DBA_ROLE compatibility role are unlocked after one minute has passed since the most recent failed login attempt.</td>
<td>Unlimited</td>
<td></td>
</tr>
<tr>
<td>Policy-option-name</td>
<td>Description</td>
<td>Default value</td>
<td>Applies to:</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>max_days_since_login</td>
<td>The maximum number of days that can elapse between two successive logins by the same user.</td>
<td>Unlimited</td>
<td>All users except those with the MANAGE ANY USER system privilege</td>
</tr>
<tr>
<td>max_non_dba_connections</td>
<td>The maximum number of concurrent connections that users can make. This option is only supported in the root login policy.</td>
<td>Unlimited</td>
<td>All users except those with the SERVER OPERATOR or DROP CONNECTION system privilege</td>
</tr>
<tr>
<td>password_life_time</td>
<td>The maximum number of days before a password must be changed.</td>
<td>Unlimited</td>
<td>All users</td>
</tr>
<tr>
<td>password_grace_time</td>
<td>The number of days before the password expires during which login is allowed, but the default post_login procedure issues warnings.</td>
<td>0</td>
<td>All users</td>
</tr>
<tr>
<td>password_expiry_on_next_login</td>
<td>If the value for this option is ON, the user's password expires after the next login.</td>
<td>OFF</td>
<td>All users</td>
</tr>
<tr>
<td>root_auto_unlock_time</td>
<td>The time period after which locked accounts are automatically unlocked. This option is only supported in the root login policy.</td>
<td>1 minute</td>
<td>Users with the MANAGE ANY USER system privilege</td>
</tr>
</tbody>
</table>

**Remarks**

If you do not specify a policy option, values for this login policy are taken from the root login policy. New policies do not inherit the MAX_NON_DBA_CONNECTIONS and ROOT_AUTO_UNLOCK_TIME policy options.
All new databases include a root login policy. You can modify the root login policy values, but you cannot delete the policy. An overview of the default values for the root login policy is provided in the table above.

**Privileges**

You must have the MANAGE ANY LOGIN POLICY system privilege.

**Side effects**

None.

**See also**

- “ALTER LOGIN POLICY statement” on page 445
- “ALTER USER statement” on page 508
- “COMMENT statement” on page 538
- “CREATE USER statement” on page 721
- “DROP LOGIN POLICY statement” on page 761
- “DROP USER statement” on page 786
- “Login policies” [SQL Anywhere Server - Database Administration]
- “Creating a new login policy” [SQL Anywhere Server - Database Administration]
- “Assigning a login policy to an existing user” [SQL Anywhere Server - Database Administration]

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Examples**

The following example creates the Test1 login policy. This example has an unlimited password life and allows the user a maximum of 5 attempts to enter a correct password before the account is locked.

```
CREATE LOGIN POLICY Test1
  PASSWORD_LIFE_TIME=UNLIMITED
  MAX_FAILED_LOGIN_ATTEMPTS=5;
```

The following example shows typical settings for a new login policy (ldap_user_policy) that uses LDAP user authentication. Both a primary and a secondary server configuration object (which would have been previously created) are specified to allow failover to the secondary LDAP server, and the ability to failover to standard authentication is allowed when system resources, network resources, or, both primary and secondary LDAP servers are unresponsive. This example provides a combination of authentication options that permits responsiveness with cached values when an LDAP server cannot keep up with incoming requests. This example assumes that the login_mode database option includes 'Standard'. You cannot paste and run this example, since the primary and secondary servers mentioned in the example are fictitious.

```
CREATE LOGIN POLICY ldap_user_policy
  LDAP_PRIMARY_SERVER=ldapsrv1
  LDAP_SECONDARY_SERVER=ldapsrv2
  LDAP_FAILOVER_TO_STD=ON;
```
CREATE MATERIALIZED VIEW statement

Creates a materialized view.

Syntax

```sql
CREATE MATERIALIZED VIEW
[ owner.]materialized-view-name [ ( alt-column-names, ... ) ]
[ IN dbspace-name ]
AS select-statement
[ CHECK { IMMEDIATE | MANUAL } REFRESH ]
```

`alt-column-names` : `( column-name [,...] )`

Parameters

**alt-column-names clause**  Use this clause to specify alternate names for the columns in the materialized view. If you specify alternate column names, the number of columns listed in `alt-column-names` must match the number of columns in `select-statement`. If you do not specify alternate column names, the names are set to those in `select-statement`.

**IN clause**  Use this clause to specify the dbspace in which to create the materialized view. If this clause is not specified, then the materialized view is created in the dbspace specified by the default_dbspace option. Otherwise, the system dbspace is used.

**AS clause**  Use this clause to specify, in the form of a SELECT statement, the data to use to populate the materialized view. A materialized view definition can only reference base tables; it cannot reference views, other materialized views, or temporary tables. `select-statement` must contain column names or have an alias name specified. If you specify `alt-column-names`, those names are used instead of the aliases specified in `select-statement`.

Column names in the SELECT statement must be specified explicitly; you cannot use the `SELECT *` construct. For example, you cannot specify `CREATE MATERIALIZED VIEW matview AS SELECT * FROM table-name`. Also, you should fully qualify objects names in the `select-statement`.

**CHECK clause**  Use this clause to validate the statement without actually creating the view. When you specify the CHECK clause:

- The database server performs the normal language checks that would be carried out if CREATE MATERIALIZED VIEW was executed without the clause, and any errors generated are returned as usual.

- The database server does not perform the actual creation of the view, so certain errors that would occur at creation time are not generated. For example, an error indicating that the specified view name already exists is not generated. This allows you to use the CHECK clause to test intended changes to the definition of the view, without a conflict with the naming of the view.

- If CHECK IMMEDIATE REFRESH is used then the database server verifies that the syntax is valid for an immediate view and returns any errors.

- No changes are made to the database, and nothing is recorded in the transaction log.
There is an implicit commit at the beginning of statement execution and a rollback at the end to release all locks obtained during execution.

See also

- “Materialized views restrictions” [SQL Anywhere Server - SQL Usage]
- “Additional dbspaces considerations” [SQL Anywhere Server - Database Administration]

Remarks

When you create a materialized view, it is a manual view and uninitialized. That is, it has a manual refresh type, and it has not been refreshed (populated with data). To initialize the view, execute a REFRESH MATERIALIZED VIEW statement, or use the sa_refresh_materialized_views system procedure.

You can encrypt a materialized view, change its PCTFREE setting, change its refresh type, and enable or disable its use by the optimizer. However, you must create the materialized view first, and then use the ALTER MATERIALIZED VIEW to change these settings. The default values for materialized views at creation time are:

- NOT ENCRYPTED
- ENABLE USE IN OPTIMIZATION
- PCTFREE is set according to the database page size: 200 bytes for a 4 KB page size, and 100 bytes for a 2 KB page size
- MANUAL REFRESH

Several database and server options must be in effect to create a materialized view.

The sa_recompile_views system procedure does not affect materialized views.

See also

- “Materialized views restrictions” [SQL Anywhere Server - SQL Usage]
- “REFRESH MATERIALIZED VIEW statement” on page 925

Privileges

You must have the CREATE MATERIALIZED VIEW system privilege to create materialized views owned by you. You must also be the owner of, or have SELECT privileges on, the underlying object referred to by the materialized view.

You must have the CREATE ANY MATERIALIZED VIEW or CREATE ANY OBJECT system privilege to create materialized views owned by others.

Side effects

While executing, the CREATE MATERIALIZED VIEW statement places exclusive locks, without blocking, on all tables referenced by the materialized view. If one of the referenced tables cannot be locked, the statement fails and an error is returned.
See also

- “Materialized views” [SQL Anywhere Server - SQL Usage]
- “Advanced: Status and properties for materialized views” [SQL Anywhere Server - SQL Usage]
- “ALTER MATERIALIZED VIEW statement” on page 450
- “DROP MATERIALIZED VIEW statement” on page 762
- “REFRESH MATERIALIZED VIEW statement” on page 925
- “CREATE VIEW statement” on page 724
- “sa_refresh_materialized_views system procedure” on page 1210

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example creates a materialized view containing confidential information about employees in the SQL Anywhere sample database. You must subsequently execute a REFRESH MATERIALIZED VIEW statement to initialize the view for use, as shown in the example.

```sql
CREATE MATERIALIZED VIEW EmployeeConfid2 AS
SELECT EmployeeID, Employees.DepartmentID,
SocialSecurityNumber, Salary, ManagerID,
Departments.DepartmentName, Departments.DepartmentHeadID
FROM GROUPO.Employees, GROUPO.Departments
WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid2;
```

CREATE MESSAGE statement [T-SQL]

Creates a message number/message string pair. The message number can be used in PRINT and RAISERROR statements.

Syntax

```sql
CREATE MESSAGE message-number AS message-text
```

Parameters

- **message-number**  The message number of the message to add. The message number for a user-defined message must be 20000 or greater.

- **message-text**  The text of the message to add. The maximum length is 255 bytes. PRINT and RAISERROR recognize placeholders in the message text. A single message can contain up to 20 unique placeholders in any order. These placeholders are replaced with the formatted contents of any arguments that follow the message when the text of the message is sent to the client.

The placeholders are numbered to allow reordering of the arguments when translating a message to a language with a different grammatical structure. A placeholder for an argument appears as "%nn!": a percent sign (%), followed by an integer from 1 to 20, followed by an exclamation mark (!), where the
integer represents the position of the argument in the argument list. "%1!" is the first argument, "%2!" is
the second argument, and so on.

There is no parameter corresponding to the language argument for sp_addmessage.

Remarks
Adds a user-defined message to the ISYSUSERMESSAGE system table for use by PRINT and
RAISERROR statements.

Privileges
You must have the CREATE MESSAGE or CREATE ANY OBJECT system privilege.

Side effects
Automatic commit.

See also

● “PRINT statement [T-SQL]” on page 918
● “RAISERROR statement” on page 920
● “DROP MESSAGE statement” on page 763
● “SYSUSERMESSAGE system view” on page 1414

Standards and compatibility

● SQL/2008 Vendor extension.

● Transact-SQL CREATE MESSAGE supplies the functionality provided by the sp_addmessage
  system procedure in Adaptive Server Enterprise.

Example
The following example creates a new message:

    CREATE MESSAGE 20000 AS 'End of line reached';

CREATE MIRROR SERVER statement

Create or replaces a mirror server that is being used for database mirroring or read-only scale-out.

Syntax 1

CREATE [ OR REPLACE ] MIRROR SERVER mirror-server-name
AS { PRIMARY | MIRROR | ARBITER | PARTNER }
[ server-option = string [ ... ] ]

Note
Read-only scale-out and database mirroring each require a separate license. See “Separately licensed
components” [SQL Anywhere 16 - Introduction].
Syntax 2

```
CREATE [ OR REPLACE ] MIRROR SERVER mirror-server-name
AS COPY
{ FROM SERVER parent-name [ OR SERVER server-name ] | USING AUTO PARENT }
[ server-option = string [ ... ] ]
```

server-option:
connection_string
logfile
preferred
state_file

parent-name:
server-name | PRIMARY

Parameters

**OR REPLACE clause**  CREATE MIRROR SERVER creates the mirror server. An error is returned if a mirror server with the specified name already exists in the database.

Specifying OR REPLACE creates a mirror server if the server does not already exist in the database, and replaces it if it does exist.

**AS clause**  You can specify one of the following server types:

- **PRIMARY**  The mirror server with type PRIMARY defines a virtual or logical server, rather than an actual database server. The name of this server is the alternate server name for the database. The alternate server name can be used by applications to connect to the server currently acting as the primary server. The connection string for the server marked as PRIMARY
  
  - Defines the connection string used by copy nodes to connect to the root node or PRIMARY parent.
  
  - Defines the connection string used by the connection parameter NodeType PRIMARY value.

  There can be only one PRIMARY server for a database.

- **MIRROR**  The mirror server with type MIRROR defines a virtual or logical server, rather than an actual database server. The name of this server is the alternate mirror server name for the database. The alternate mirror server name can be used by applications to connect to the server currently acting as the read-only mirror. The server marked as MIRROR also defines the connection string used by the NodeType connection parameter MIRROR value. There can be only one MIRROR server for a database.

- **ARBITER**  In a database mirroring system, the arbiter server assists in determining which of the PARTNER servers takes ownership of the database. The arbiter server must be defined with a connection string that can be used by the partner servers to connect to the arbiter. There can be only one ARBITER server for a database.

- **PARTNER**  The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.
In a database mirroring system, the partners use the connection string value to connect to each other. In a read-only scale-out system, the connection string is used by a copy-node that has this server as its parent.

- **Mirroring or mirroring with read-only scale-out** You must define two PARTNER servers for database mirroring, and both must have a connection string and a state file.

  In a database mirroring system, servers defined as PARTNER are eligible to become the primary server and take ownership of the database.

- **Read-only scale-out without mirroring** You must define one PARTNER server for read-only scale-out, and it must have a connection string and no state file. This server is the root server, and runs the only copy of the database that allows both read and write operations.

  - **COPY** In a read-only scale-out system, this value specifies that the database server is a copy node. All connections to the database on this server are read-only. The name of the mirror server must correspond to the name of the database server, as specified by the -n server option, and must match the value of the SERVER connection string parameter specified in the connection_string mirror server option.

    When AS COPY is specified, then you must also specify either the FROM SERVER or USING AUTO PARENT clause.

    The connection string is used by the NodeType connection parameter COPY value and it is also used by other copy nodes that have this server as their parent.

    When adding copy nodes to a read-only scale-out system, you can either execute a CREATE MIRROR SERVER statement for the copy node, or have the root server define the mirror server automatically.

  **FROM SERVER clause** This clause can only be used when AS COPY is specified. This clause constructs a tree of servers for a scale-out system and indicates which servers the copy nodes obtain transaction log pages from.

    The parent can be specified using the name of the mirror server or PRIMARY. An alternate parent for the copy node can be specified using the OR SERVER clause.

    In a database mirroring system that has only two levels (partner and copy nodes), the copy nodes obtain transaction log pages from the current primary or mirror server.

    A copy node determines which server to connect to by using its mirror server definition that is stored in the database. From its definition, it can locate the definition of its parent, and from its parent’s definition, it can obtain the connection string to connect to the parent.

    You do not have to explicitly define copy nodes for the scale-out system: you can choose to have the root node define the copy nodes when they connect.

  **OR SERVER clause** Use the OR SERVER clause to specify an alternate parent for the copy node.

  **USING AUTO PARENT clause** This clause can only be used when AS COPY is specified. This clause causes the primary server to assign a parent for this server. When you use this clause to replace an
existing copy node server, the definitions for the parent and alternate parent for the copy node do not change.

**server-option clause**  The following options are supported:

- **connection_string server option**  Specifies the connection string to be used to connect to the server. The connection string for a mirror server should not include a user ID or password because they are not used when one mirror server connects to another mirror server.

  For a list of connection parameters, see “Connection parameters” [SQL Anywhere Server - Database Administration].

- **logfile server option**  Specifies the location of the file that contains one line per request that is sent between mirror servers if database mirroring is used. This file is used only for debugging.

- **preferred server option**  Specifies whether the server is the preferred server in the mirroring system. You can specify either YES or NO. The preferred server assumes the role of primary server whenever possible. You specify this option when defining PARTNER servers.

- **state_file server option**  Specifies the location of the file used for maintaining state information about the mirroring system. This option is required for database mirroring. In a mirroring system, a state file must be specified for servers with type PARTNER. For arbiter servers, the location is specified as part of the command to start the server.

**Remarks**

This statement creates or replaces a mirror server definition; it does not change a mirror server definition. To change a mirror server definition, use the ALTER MIRROR SERVER statement.

In a database mirroring system, the mirror server type can be PRIMARY, MIRROR, ARBITER, or PARTNER.

In a read-only scale-out system, the mirror server type can be PRIMARY, PARTNER, or COPY.

Mirror server names for servers of type PARTNER or COPY must match the names of the database servers that are part of the mirroring system (the name used with the -n server option). This requirement allows each database server to find its own definition and that of its parent. Also, all copy node servers must have unique server names. mirror-server-name, parent-name, and server-name above must be 7-bit ASCII characters.

To use a copy node as the arbiter for the database it is copying in a database mirroring system, create the arbiter server with a name that does not match the server name of any of the database servers in the high availability system. In this configuration, the name of the arbiter is used as a placeholder in the mirror server definition to hold the connection string for the arbiter.

**Privileges**

You must have the MANAGE ANY MIRROR SERVER system privilege.

**Side effects**

Automatic commit.
Standards and compatibility

- SQL/2008  Vendor extension

Example

The following statement creates a mirror server that can be used as the primary server in a database mirroring system:

```sql
CREATE MIRROR SERVER "scaleout_primary"
AS PRIMARY
connection_string = 'server=scaleout_primary;host=winxp-2:6871,winxp-3:6872';
```

The following statement creates a mirror server that can be used as the mirror server in a database mirroring system:

```sql
CREATE MIRROR SERVER "scaleout_mirror"
AS MIRROR
connection_string = 'server=scaleout_mirror;host=winxp-2:6871,winxp-3:6872';
```

The following statement creates a mirror server that can be used as the arbiter in a database mirroring system:

```sql
CREATE MIRROR SERVER "scaleout_arbiter"
AS ARBITER
connection_string = 'server=scaleout_arbiter;host=winxp-4:6870';
```

The following statement creates two mirror servers that can be used as partners server in a database mirroring system:

```sql
CREATE MIRROR SERVER "scaleout_server1"
AS PARTNER
connection_string = 'server=scaleout_server1;HOST=winxp-2:6871'
state_file = 'c:\\server1\\server1.state';

CREATE MIRROR SERVER "scaleout_server2"
AS PARTNER
connection_string = 'server=scaleout_server2;HOST=winxp-3:6872'
state_file = 'c:\\server2\\server2.state';
```
The following statement creates a copy node that can act as the arbiter in a database mirroring system:

```sql
CREATE MIRROR SERVER "scaleout_child"
AS COPY FROM SERVER "scaleout_primary"
connection_string = 'server=scaleout_child;host=winxp-5:6878';
```

The following statement defines a copy node as the arbiter for a different database mirroring system:

```sql
CREATE MIRROR SERVER "The Arbiter"
AS ARBITER
connection_string = 'server=scaleout_child;host=winxp-5:6878';
```

The following statement preserves the current parent if `server-name` already exists. However, it does not auto-generate a new parent.

```sql
CREATE OR REPLACE MIRROR SERVER "server-name" AS COPY USING AUTO PARENT;
```

## CREATE PROCEDURE statement [External call]

Creates an interface to a native or external procedure.

### Syntax

```sql
CREATE [ OR REPLACE ] PROCEDURE [ owner. ] procedure-name
    ( [ [ parameter[, ... ] ] ]
    [ SQL SECURITY { INVOKER | DEFINER } ]
    [ RESULT ( result-column[, ... ] ) | NO RESULT SET ]
    [ DYNAMIC RESULT SETS integer-expression ]
    [ EXTERNAL NAME 'native-call'
        EXTERNAL NAME 'c-call' LANGUAGE { C_ESQL32 | C_ESQL64 | C_ODBC32 | C_ODBC64 }
        EXTERNAL NAME 'clr-call' LANGUAGE CLR
        EXTERNAL NAME 'perl-call' LANGUAGE PERL
        EXTERNAL NAME 'php-call' LANGUAGE PHP
        EXTERNAL NAME 'java-call' LANGUAGE JAVA ]

    parameter :
        [ parameter-mode ] parameter-name data-type [ DEFAULT expression ]
        SQLCODE
        SQLSTATE

    parameter-mode :
        IN
        OUT
        INOUT

    native-call :
        [ operating-system ] function-name@library

    result-column :
    column-name data-type

    c-call :
        [ operating-system ] function-name@library; ...

    clr-call :
        dll-name::function-name( param-type-1[, ... ] )
```

620 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
perl-call:
<file=perl-file> $sa_perl_return = perl-subroutine( $sa_perl_arg0[, ... ] )

php-call:
<file=php-file> print php-func( $argv[1][, ... ] )

java-call:
[package-name.]class-name.method-name method-signature

operating-system:
Unix

method-signature:
( [ field-descriptor, ... ] ) return-descriptor

field-descriptor and return-descriptor:
{ Z
 | B
 | S
 | J
 | F
 | D
 | C
 | V
 | [descriptor
 | Lclass-name;
 }

Parameters
You can create permanent stored procedures that call external or native procedures written in a variety of
programming languages. You can use PROC as a synonym for PROCEDURE.

OR REPLACE clause Specifying OR REPLACE creates a new procedure, or replaces an existing
procedure with the same name. This clause changes the definition of the procedure, but preserves existing
privileges. An error is returned if you attempt to replace a procedure that is already in use.

Parameters Parameter names must conform to the rules for other database identifiers such as column
names. They must be a valid SQL data type.

Parameters can be prefixed with one of the keywords IN, OUT, or INOUT. If you do not specify one of
these values, parameters are INOUT by default. The keywords have the following meanings:

○ IN The parameter is an expression that provides a value to the procedure.

○ OUT The parameter is a variable that could be given a value by the procedure.

○ INOUT The parameter is a variable that provides a value to the procedure, and could be given a new
value by the procedure.

When procedures are executed using the CALL statement, not all parameters need to be specified. If a
default value is provided in the CREATE PROCEDURE statement, missing parameters are assigned the
default values. If an argument is not provided in the CALL statement, and no default is set, an error is
given.
SQLSTATE and SQLCODE are special OUT parameters that output the SQLSTATE or SQLCODE value when the procedure ends. The SQLSTATE and SQLCODE special values can be checked immediately after a procedure call to test the return status of the procedure.

The SQLSTATE and SQLCODE special values are modified by the next SQL statement. Providing SQLSTATE or SQLCODE as procedure arguments allows the return code to be stored in a variable.

Specifying OR REPLACE (CREATE OR REPLACE PROCEDURE) creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

You cannot create TEMPORARY external call procedures.

**RESULT clause** The RESULT clause declares the number and type of columns in the result set. The parenthesized list following the RESULT keyword defines the result column names and types. This information is returned by the embedded SQL DESCRIBE or by ODBC SQLDescribeCol when a CALL statement is being described.

Embedded SQL (LANGUAGE C_ESQL32, LANGUAGE C_ESQL64) or ODBC (LANGUAGE C_ODBC32, LANGUAGE C_ODBC64) external procedures can return 0 or 1 result sets.

Perl or PHP (LANGUAGE PERL, LANGUAGE PHP) external procedures cannot return result sets. Procedures that call native functions loaded by the database server cannot return result sets.

CLR or Java (LANGUAGE CLR, LANGUAGE JAVA) external procedures can return 0, 1, or more result sets.

Some procedures can produce more than one result set, with different numbers of columns, depending on how they are executed. For example, the following procedure returns two columns under some circumstances, and one in others.

```
CREATE PROCEDURE names(IN formal char(1))
BEGIN
  IF formal = 'n' THEN
    SELECT GivenName
    FROM GROUPO.Employees
  ELSE
    SELECT Surname, GivenName
    FROM Employees
  END IF
END;
```

Procedures with variable result sets must be written without a RESULT clause, or in Transact-SQL. Their use is subject to the following limitations:

- **Embedded SQL** You must DESCRIBE the procedure call after the cursor for the result set is opened, but before any rows are returned, to get the proper shape of result set. The CURSOR cursor-name clause on the DESCRIBE statement is required.

- **ODBC, OLE DB, ADO.NET** Variable result-set procedures can be used by applications using these interfaces. The proper description of the result sets is carried out by the driver or provider.
○ **Open Client applications**  Variable result-set procedures can be used by Open Client applications.

If your procedure returns only one result set, you should use a RESULT clause. The presence of this clause prevents ODBC and Open Client applications from re-describing the result set after a cursor is open.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure's defined result set. Therefore, ODBC does not always describe column names as defined in the RESULT clause of the procedure definition. To avoid this problem, use column aliases in the SELECT statement that generates the result set.

**NO RESULT SET clause**  Declares that no result set is returned by this procedure. This declaration can lead to a performance improvement.

**DYNAMIC RESULT SETS clause**  Use this clause with LANGUAGE CLR and LANGUAGE JAVA calls. The DYNAMIC RESULT SETS clause is used to specify the number of dynamic result sets that will be returned by the procedure. When a RESULT clause is specified and the DYNAMIC RESULT SETS clause is not specified, it is assumed that the number of dynamic result sets is 1. When neither the RESULT clause nor the DYNAMIC RESULT SETS clause is specified, no result set is expected and an error will result if a result set is generated.

The C_ESQL32, C_ESQL64, C_ODBC32, and C_ODBC64 external environments can also return result sets (like CLR and JAVA), but they are restricted to only one dynamic result set.

Procedures that call into Perl or PHP (LANGUAGE PERL, LANGUAGE PHP) external functions cannot return result sets. Procedures that call native functions loaded by the database server cannot return result sets.

**SQL SECURITY clause**  The SQL SECURITY clause defines whether the procedure is executed as the INVOKER (the user who is calling the procedure), or as the DEFINER (the user who owns the procedure). The default is DEFINER. For external calls, this clause establishes the ownership context for unqualified object references in the external environment.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. Also, when SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, care should be taken to qualify all object names (tables, procedures, and so on) with their appropriate owner. For example, suppose user1 creates the following procedure:

```sql
CREATE PROCEDURE user1.myProcedure()
RESULT( columnA INT )
SQL SECURITY INVOKER
BEGIN
  SELECT columnA FROM table1;
END;
```

If user2 attempts to run this procedure and a table user2.table1 does not exist, a table lookup error results. Additionally, if a user2.table1 does exist, that table is used instead of the intended user1.table1. To prevent this situation, qualify the table reference in the statement (user1.table1, instead of just table1).
**EXTERNAL NAME clause**  A procedure using the EXTERNAL NAME clause with no LANGUAGE attribute defines an interface to a native function written in a programming language such as C. The native function is loaded by the database server into its address space.

The **library** name can include the file extension, which is typically .dll on Windows and .so on Unix. In the absence of the extension, the software appends the platform-specific default file extension for libraries. The following is a formal example.

```sql
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@mylib.dll;Unix:mystring@mylib.so';
```

A simpler way to write the above EXTERNAL NAME clause, using platform-specific defaults, is as follows:

```sql
CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
EXTERNAL NAME 'mystring@mylib';
```

When called, the library containing the function is loaded into the address space of the database server. The native function will execute as part of the server. In this case, if the function causes a fault, then the database server will be terminated. Because of this, loading and executing functions in an external environment using the LANGUAGE attribute is recommended. If a function causes a fault in an external environment, the database server will continue to run.

For syntaxes that support **operating-system**, if you do not specify **operating-system**, then it is assumed that the procedure runs on all platforms. If you specify **Unix** for one of the calls, then it is assumed that the other call is for Windows.

- **EXTERNAL NAME 'c-call' LANGUAGE { C_ESQL32 | C_ESQL64 | C_ODBC32 | C_ODBC64 } clause**  To call a compiled native C function in an external environment instead of within the database server, the stored procedure or function is defined with the EXTERNAL NAME clause followed by the LANGUAGE attribute specifying one of C_ESQL32, C_ESQL64, C_ODBC32, or C_ODBC64.

  When the LANGUAGE attribute is specified, then the library containing the function is loaded by an external process and the external function will execute as part of that external process. In this case, if the function causes a fault, then the database server will continue to run.

  The following is a sample procedure definition.

  ```sql
  CREATE PROCEDURE ODBCinsert(
      IN ProductName CHAR(30),
      IN ProductDescription CHAR(50)
  )
  NO RESULT SET
  EXTERNAL NAME 'ODBCexternalInsert@extodbc.dll'
  LANGUAGE C_ODBC32;
  ```

- **EXTERNAL NAME clr-call LANGUAGE CLR clause**  To call a .NET function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE CLR attribute.

  A CLR stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in a .NET language such as C# or Visual Basic.
and the execution of the procedure or function takes place outside the database server (that is, within a separate .NET executable).

```sql
CREATE PROCEDURE clr_interface(
    IN p1 INT,
    IN p2 UNSIGNED SMALLINT,
    OUT p3 LONG VARCHAR)
NO RESULT SET
EXTERNAL NAME 'CLRlib.dll::CLRproc.Run( int, ushort, out string )'
LANGUAGE CLR;
```

- **EXTERNAL NAME perl-call LANGUAGE PERL clause**  To call a Perl function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PERL attribute.

A Perl stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Perl and the execution of the procedure or function takes place outside the database server (that is, within a Perl executable instance).

The following is a sample procedure definition.

```sql
CREATE PROCEDURE PerlWriteToConsole( IN str LONG VARCHAR)
NO RESULT SET
EXTERNAL NAME '<file=PerlConsoleExample>WriteToServerConsole( $sa_perl_arg0 )'
LANGUAGE PERL;
```

- **EXTERNAL NAME php-call LANGUAGE PHP clause**  To call a PHP function in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE PHP attribute.

A PHP stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in PHP and the execution of the procedure or function takes place outside the database server (that is, within a PHP executable instance).

The following is a sample procedure definition.

```sql
CREATE PROCEDURE PHPPopulateTable()
NO RESULT SET
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()' LANGUAGE PHP;
```

- **EXTERNAL NAME java-call LANGUAGE JAVA clause**  To call a Java method in an external environment, the procedure interface is defined with an EXTERNAL NAME clause followed by the LANGUAGE JAVA attribute.

A Java-interfacing stored procedure or function behaves the same as a SQL stored procedure or function except that the code for the procedure or function is written in Java and the execution of the procedure or function takes place outside the database server (that is, within a Java VM).

The following is a sample procedure definition.

```sql
CREATE PROCEDURE HelloDemo( IN name LONG VARCHAR )
NO RESULT SET
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```
The descriptors for arguments and return values from Java methods have the following meanings:

<table>
<thead>
<tr>
<th>Field type</th>
<th>Java data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>byte</td>
</tr>
<tr>
<td>C</td>
<td>char</td>
</tr>
<tr>
<td>D</td>
<td>double</td>
</tr>
<tr>
<td>F</td>
<td>float</td>
</tr>
<tr>
<td>I</td>
<td>int</td>
</tr>
<tr>
<td>J</td>
<td>long</td>
</tr>
<tr>
<td>L class-name;</td>
<td>An instance of the class class-name. The class name must be fully qualified, and any dot in the name must be replaced by a /. For example, java/lang/String.</td>
</tr>
<tr>
<td>S</td>
<td>short</td>
</tr>
<tr>
<td>V</td>
<td>void</td>
</tr>
<tr>
<td>Z</td>
<td>Boolean</td>
</tr>
<tr>
<td>[</td>
<td>Use one for each dimension of an array.</td>
</tr>
</tbody>
</table>

Remarks

The CREATE PROCEDURE statement creates a procedure in the database. You can create procedures for other users by specifying an owner. A procedure is invoked with a CALL statement.

If a stored procedure returns a result set, it cannot also set output parameters or return a return value.

When referencing a temporary table from multiple procedures, a potential issue can arise if the temporary table definitions are inconsistent and statements referencing the table are cached.

Privileges

You must have the CREATE PROCEDURE system privilege to create external procedures you own.

You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create external procedures owned by others.

Side effects

Automatic commit.
See also

- “References to temporary tables within procedures” [SQL Anywhere Server - SQL Usage]
- “ALTER PROCEDURE statement” on page 456
- “CALL statement” on page 530
- “CREATE FUNCTION statement” on page 594
- “CREATE FUNCTION statement [External call]” on page 580
- “CREATE PROCEDURE statement” on page 639
- “CREATE PROCEDURE statement [Web service]” on page 628
- “CREATE PROCEDURE statement [T-SQL]” on page 637
- “DROP PROCEDURE statement” on page 765
- “GRANT statement” on page 827
- “SQL data types” on page 89
- “The ESQL and ODBC external environments” [SQL Anywhere Server - Programming]
- “SQL Anywhere external call interface” [SQL Anywhere Server - Programming]
- “SQL Anywhere external environment support” [SQL Anywhere Server - Programming]
- “The PERL external environment” [SQL Anywhere Server - Programming]
- “The CLR external environment” [SQL Anywhere Server - Programming]
- “The PHP external environment” [SQL Anywhere Server - Programming]
- “The Java external environment” [SQL Anywhere Server - Programming]

Standards and compatibility

- **SQL/2008**  
  CREATE PROCEDURE for an external language environment is a core feature of the SQL/2008 standard, though some of its components supported in SQL Anywhere are optional SQL/2008 language features. A subset of these features include:

  - The SQL SECURITY clause is SQL/2008 optional language feature T324.

  - The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to an external procedure is SQL/2008 language feature T041.

  - The ability to create or modify a schema object within an external procedure, using statements such as CREATE TABLE or DROP TRIGGER, is SQL/2008 language feature T653.

  - The ability to use a dynamic-SQL statement within an external procedure, including statements such as CONNECT, EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is SQL/2008 language feature T654.

  - JAVA external procedures embody SQL/2008 language feature J621.

Several clauses of the CREATE PROCEDURE statement are vendor extensions. These include:

  - Support for C_ESQL32, C_ESQL64, C_ODBC32, C_ODBC64, CLR, PERL, and PHP in the LANGUAGES clause are vendor extensions. The SQL/2008 standard supports "C" as an environment-name as optional language feature B122.

  - The format of external-call is implementation-defined.

  - The RESULT and NO RESULT SET clauses are vendor extensions. The SQL/2008 standard uses the RETURNS clause.
The optional DEFAULT clause for a specific routine parameter is a vendor extension.

The optional OR REPLACE clause is a vendor extension.

- **Transact-SQL** CREATE PROCEDURE for an external routine is supported by Adaptive Server Enterprise. Adaptive Server Enterprise supports C-language and Java language external routines.

## CREATE PROCEDURE statement [Web service]

Creates a user-defined web client procedure that makes HTTP or SOAP requests to an HTTP server.

### Syntax

```
CREATE [ OR REPLACE ] PROCEDURE [ owner.]procedure-name ( [ parameter, ... ] )
[ RESULT ( attribute-column-name datatype, value-column-name datatype ) ]
URL url-string
[ TYPE { http-type-spec-string | soap-type-spec-string } ]
[ HEADER header-string ]
[ CERTIFICATE certificate-string ]
[ CLIENTPORT clientport-string ]
[ PROXY proxy-string ]
[ SET protocol-option-string ]
[ SOAPHEADER soap-header-string ]
[ NAMESPACE namespace-string ]
```

- **http-type-spec-string**:
  ```
  HTTP: { GET
  [ POST[:MIME-type] ]
  [ PUT[:MIME-type] ]
  [ DELETE
  [ HEAD ] ]
  ```

- **soap-type-spec-string**:
  ```
  SOAP: { RPC | DOC };
  ```

- **parameter**:
  ```
  parameter-mode parameter-name data-type [ DEFAULT expression ]
  ```

- **parameter-mode**:
  ```
  IN
  | OUT
  | INOUT
  ```

- **url-string**:
  ```
  { HTTP | HTTPS | HTTPS_FIPS }://[user:password@]hostname[:port] [/path]
  ```

- **protocol-option-string**:
  ```
  [ http-option-list ]
  [, soap-option-list ]
  ```

- **http-option-list**:
  ```
  HTTP(
  [ CH(UNK)= { ON | OFF | AUTO } ]
  [ ; VER[SION]= { 1.0 | 1.1 } ] ; kto=number-of-seconds ]
  ```
Parameters

You can create or replace a web services client procedure. You can use PROC as a synonym for PROCEDURE.

For SOAP requests, the procedure name is used as the SOAP operation name by default. For more information, see the SET clause.

Note

You cannot create TEMPORARY web services procedures.

OR REPLACE clause

Specifying OR REPLACE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

Parameters

Parameter names must conform to the rules for other database identifiers such as column names. They must be a valid SQL data type.

Only SOAP requests support the transmission of typed data such as FLOAT, INT, and so on. HTTP requests support the transmission of strings only, so you are limited to CHAR types.

When procedures are executed using the CALL statement, not all parameters need to be specified. If a default value is provided in the CREATE PROCEDURE statement, missing parameters are assigned the default values. If an argument is not provided in the CALL statement, and no default is set, an error is given.

Parameters can be prefixed with one of the keywords IN, OUT, or INOUT. If you do not specify one of these values, parameters are INOUT by default. The keywords have the following meanings:

- **IN** The parameter is an expression that provides a value to the procedure.
- **OUT** The parameter is a variable that could be given a value by the procedure.
- **INOUT** The parameter is a variable that provides a value to the procedure, and could be given a new value by the procedure.

RESULT clause

The RESULT clause is required to use the procedure in a SELECT statement. The RESULT clause must return two columns. The first column contains HTTP response header, status, and response body attributes, while the second column contains the values for these attributes. The RESULT clause must specify two character data types. For example, VARCHAR or LONG VARCHAR. If the RESULT clause is not specified, the default column names are Attribute and Value and their data types are LONG VARCHAR.

URL clause

Specifies the URI of the web service. The optional user name and password parameters provide a means of supplying the credentials needed for HTTP basic authentication. HTTP basic
authentication base-64 encodes the user and password information and passes it in the Authentication header of the HTTP request. When specified in this way, the user name and password are passed unencrypted, as part of the URL.

For procedures of type HTTP:GET, query parameters can be specified within the URL clause in addition to being automatically generated from parameters passed to a procedure.

```
URL 'http://localhost/service?parm=1
```

**TYPE clause**  Specifies the format used when making the web service request. SOAP:RPC is used when SOAP is specified or no TYPE clause is included. HTTP:POST is used when HTTP is specified.

The TYPE clause allows the specification of a MIME-type for HTTP:GET, HTTP:POST, and HTTP:PUT types. The *MIME-type* specification is used to set the Content-Type request header and set the mode of operation to allow only a single call parameter to populate the body of the request. Only zero or one parameter may remain when making a web service stored procedure call after parameter substitutions have been processed. Calling a web service procedure with a NULL value or no parameter (after substitutions) results in a request with no body and a content-length of zero. Parameter names and values (multiple parameters are permitted) are URL encoded within the body of the HTTP request.

Some typical MIME-types include:

- text/plain
- text/html
- text/xml

The keywords for the TYPE clause have the following meanings:

- **'HTTP:GET’**  By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the URL.

  For example, the following request is produced when a client submits a request from the URL, http://localhost/WebServiceName?arg1=param1&arg2=param2:

  ```
  GET /WebServiceName?arg1=param1&arg2=param2 HTTP/1.1
  // <End of Request - NO BODY>
  ```

- **'HTTP:POST’**  By default, this type uses the application/x-www-form-urlencoded MIME-type for encoding parameters specified in the body of a POST request. URL parameters are stored in the body of the request.

  For example, the following request is produced when a client submits a request the URL, http://localhost/WebServiceName?arg1=param1&arg2=param2:

  ```
  POST /WebServiceName HTTP/1.1
  Content-Type: application/x-www-form-urlencoded
  Content-Length: 19
  arg1=param1&arg2=param2
  // <End of Request>
  ```

- **HTTP:PUT**  HTTP:PUT is similar to HTTP:POST, but the HTTP:PUT type does not have a default media type.
The following example demonstrates how to configure a general purpose client procedure that uploads data to a SQL Anywhere server running the `%SQLANYSAMP16%\SQLAnywhere\HTTP\put_data.sql` sample:

```sql
ALTER PROCEDURE CPUT("data" LONG VARCHAR, resnm LONG VARCHAR, mediatype LONG VARCHAR)  
  URL 'http://localhost/resource/!resnm'  
  TYPE 'HTTP:PUT:!mediatype';  
CALL CPUT('hello world', 'hello', 'text/plain' );
```

- **HTTP:DELETE**  A web service client procedure can be configured to delete a resource located on a server. Specifying the media type is optional.

The following example demonstrates how to configure a general purpose client procedure that deletes a resource from a SQL Anywhere server running the `put_data.sql` sample:

```sql
ALTER PROCEDURE CDEL(resnm LONG VARCHAR, mediatype LONG VARCHAR)  
  URL 'http://localhost/resource/!resnm'  
  TYPE 'HTTP:DELETE:!mediatype';  
CALL CDEL('hello', 'text/plain' );
```

- **HTTP:HEAD**  The head method is identical to a GET method but the server does not return a body. A media type can be specified.

```sql
ALTER PROCEDURE CHEAD(resnm LONG VARCHAR)  
  URL 'http://localhost/resource/!resnm'  
  TYPE 'HTTP:HEAD';  
CALL CHEAD( 'hello' );
```

- **'SOAP:RPC'**  This type sets the Content-Type header to 'text/xml'. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.

- **'SOAP:DOC'**  This type sets the Content-Type header to 'text/xml'. It is similar to the SOAP:RPC type but allows you to send richer data types. SOAP operations and parameters are encapsulated in SOAP envelope XML documents.

Specifying a MIME-type for the TYPE clause automatically sets the Content-Type header to that MIME-type.

**HEADER clause**  When creating HTTP web service client procedures, use this clause to add, modify, or delete HTTP request header entries. The specification of headers closely resembles the format specified in RFC2616 Hypertext Transfer Protocol—HTTP/1.1, and RFC822 Standard for ARPA Internet Text Messages, including the fact that only printable ASCII characters can be specified for HTTP headers, and they are case-insensitive.

Headers can be defined as `header-name:value-name` pairs. Each header must be delimited from its value with a colon (:) and therefore cannot contain a colon. You can define multiple headers by delimiting each pair with `\n`, `\x0d\n`, `<LF>` (line feed), or `<CR><LF>` (carriage return followed by a line feed).

Multiple contiguous white spaces within the header are converted to a single white space.
CERTIFICATE clause  To make a secure (HTTPS) request, a client must have access to the certificate used by the HTTPS server. The necessary information is specified in a string of semicolon-separated key/value pairs. You can use the file key to specify the file name of the certificate, or you can use the certificate key to specify the server certificate in a string. You cannot specify a file and certificate key together. The following keys are available:

<table>
<thead>
<tr>
<th>Key</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td></td>
<td>The file name of the certificate.</td>
</tr>
<tr>
<td>certificate</td>
<td>cert</td>
<td>The certificate itself.</td>
</tr>
<tr>
<td>certificate_name</td>
<td>cert_name</td>
<td>The name of a certificate stored in the database.</td>
</tr>
<tr>
<td>company</td>
<td>co</td>
<td>The company specified in the certificate.</td>
</tr>
<tr>
<td>unit</td>
<td></td>
<td>The company unit specified in the certificate.</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>The common name specified in the certificate.</td>
</tr>
</tbody>
</table>

Certificates are required only for requests that are either directed to an HTTPS server, or can be redirected from a non-secure to a secure server. Only PEM formatted certificates are supported.

CLIENTPORT clause  Identifies the port number on which the HTTP client procedure communicates using TCP/IP. It is provided for and recommended only for connections through firewalls that filter "outgoing" TCP/IP connections. You can specify a single port number, ranges of port numbers, or a combination of both; for example, CLIENTPORT '85,90-97'.

PROXY clause  Specifies the URI of a proxy server. For use when the client must access the network through a proxy. Indicates that the procedure is to connect to the proxy server and send the request to the web service through it.

SET clause  Specifies protocol-specific behavior options for HTTP and SOAP. The following list describes the supported SET options. CHUNK, VERSION, REDIR and kto apply to the HTTP protocol, and OPERATION applies to the SOAP protocol.

- `HTTP(CHUNK=option)`  (HTTP or SOAP) This option allows you to specify whether to use chunking. Chunking allows HTTP messages to be broken up into several parts. Possible values are ON (always chunk), OFF (never chunk), and AUTO (chunk only if the contents, excluding auto-generated markup, exceeds 8196 bytes). For example, the following SET clause enables chunking:

  ```sql
  SET 'HTTP (CHUNK=ON)'
  ```

  If the CHUNK option is not specified, the default behavior is AUTO. If a chunked request fails in AUTO mode with a status of 505 HTTP Version Not Supported, or with 501 Not Implemented, or with 411 Length Required, the client retries the request without chunked transfer-coding.
Set the CHUNK option to OFF (never chunk) if the HTTP server does not support chunked transfer-coded requests.

Since CHUNK mode is a transfer encoding supported starting in HTTP version 1.1, setting CHUNK to ON requires that the version (VER) be set to 1.1, or not be set at all, in which case 1.1 is used as the default version.

- `HTTP(VER[SION]=ver;kto=number-of-seconds)` (HTTP or SOAP) This option allows you to specify the version of HTTP protocol that is used for the format of the HTTP message. For example, the following SET clause sets the HTTP version to 1.1:

  ```
  SET 'HTTP(VERSION=1.1)
  ```

Possible values are 1.0 and 1.1. If VERSION is not specified:

- if CHUNK is set to ON, 1.1 is used as the HTTP version
- if CHUNK is set to OFF, 1.0 is used as the HTTP version
- if CHUNK is set to AUTO, either 1.0 or 1.1 is used, depending on whether the client is sending in CHUNK mode

- `kto=number-of-seconds` Specifies the keep-alive timeout criteria (`kto`), permitting a web client procedure to instantiate and cache a keep-alive HTTP/HTTPS connection for a period of time. To cache an HTTP keep-alive connection, the HTTP version must be set to 1.1 and kto set to a non-zero value. kto may be useful, for HTTPS connections particularly, if you notice a significant performance difference between HTTP and HTTPS connections. A database connection can only cache a single keep-alive HTTP connection. Subsequent calls to a web client procedure using the same URI reuse the keep-alive connection. Therefore, the executing web client call must have a URI whose scheme, destination host and port match that of the cached URI, and the HEADER clause must not specify Connection: close. When kto is not specified, or is set to zero, HTTP/HTTPS connections are not cached.

- `REDIR(COUNT=count, STATUS=status-list)` (HTTP or SOAP) The HTTP response status codes such as 302 Found and 303 See Other are used to redirect web applications to a new URI, particularly after an HTTP POST has been performed. For example, a client request could be:

  ```
  GET /people/alice HTTP/1.1
  Host: www.example.com
  Accept: text/html, application/xhtml+xml
  Accept-Language: en, de
  ```

The web server response could be:

  ```
  HTTP/1.1 302 Found
  Location: http://www.example.com/people/alice.en.html
  ```

In response, the client would send another HTTP request to the new URI. The REDIR option allows you to control the maximum number of redirections allowed and which HTTP response status codes to automatically redirect.
For example, SET 'REDIR(count=3, status=301,307)' allows a maximum limit of 3 redirections and permits redirection for 301 and 307 statuses. If one of the other redirection status codes such as 302 or 303 is received, an error is issued (SQLCODE -983).

The default redirection limit \textit{count} is 5. By default, an HTTP client procedure will automatically redirect in response to all HTTP redirection status codes (301, 302, 303, 307). To disallow all redirection status codes, use SET \texttt{REDIR(COUNT=0)}. In this mode, a redirection response does not result in an error (SQLCODE -983). Instead, a result set is returned with the HTTP status and response headers. This permits a caller to conditionally reissue the request based on the URI contained in the \texttt{Location} header.

A web service procedure specifying a POST HTTP method which receives a 303 See Other status will issue a redirect request using the GET HTTP method.

The \texttt{Location} header can contain either an absolute path or a relative path. The HTTP client procedure will handle either. The header can also include query parameters and these will be forwarded to the redirected location. For example, if the header contained parameters such as the following, the subsequent GET or a POST will include these parameters.

\begin{verbatim}
Location: alternate_service?a=1&b=2
\end{verbatim}

In the above example, the query parameters are \texttt{a=1\&b=2}.

- \texttt{SOAP(OP[ERATION]=soap-operation-name)} (SOAP only) This option allows you to specify the name of the SOAP operation, if it is different from the name of the procedure you are creating. The value of \texttt{OPERATION} is analogous to the name of a remote procedure call. For example, if you wanted to create a procedure called accounts_login that calls a SOAP operation called login, you would specify something like the following:

\begin{verbatim}
CREATE PROCEDURE accounts_login(
    name LONG VARCHAR,
    pwd LONG VARCHAR )
SET 'SOAP (OPERATION=login)'
\end{verbatim}

If the \texttt{OPERATION} option is not specified, the name of the SOAP operation must match the name of the procedure you are creating.

The following statement shows how several \textit{protocol-option} settings are combined in the same SET clause:

\begin{verbatim}
CREATE PROCEDURE accounts_login(
    name LONG VARCHAR,
    pwd LONG VARCHAR )
SET 'HTTP ( CHUNK=ON; VERSION=1.1 ), SOAP( OPERATION=login )'
...
\end{verbatim}

\textbf{SOAPHEADER clause} (SOAP format only) When declaring a SOAP web service as a procedure, use this clause to specify one or more SOAP request header entries. A SOAP header can be declared as a static constant, or can be dynamically set using the parameter substitution mechanism (declaring IN, OUT, or INOUT parameters for \texttt{hd1}, \texttt{hd2}, and so on). A web service procedure can define one or more IN mode substitution parameters, and a single INOUT or OUT substitution parameter.
The following example illustrates how a client can specify the sending of several header entries using parameter substitution and receiving the response SOAP header data:

```
CREATE PROCEDURE soap_client
    (INOUT hd1 LONG VARCHAR, IN hd2 LONG VARCHAR, IN hd3 LONG VARCHAR)
    URL 'localhost/some_endpoint'
    SOAPHEADER '!hd1!hd2!hd3';
```

**NAMESPACE clause**  (SOAP format only) This clause identifies the method namespace usually required for both SOAP:RPC and SOAP:DOC requests. The SOAP server handling the request uses this namespace to interpret the names of the entities in the SOAP request message body. The namespace can be obtained from the WSDL (Web Services Description Language) of the SOAP service available from the web service server. The default value is the procedure's URL, up to but not including the optional path component.

**Remarks**

Parameter values are passed as part of the request. The syntax used depends on the type of request. For HTTP:GET, the parameters are passed as part of the URL; for HTTP:POST requests, the values are placed in the body of the request. Parameters to SOAP requests are always bundled in the request body.

**Privileges**

You must have the CREATE PROCEDURE system privilege to create procedures owned by you. You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege to create procedures owned by others.

**Side effects**

Automatic commit.
See also

- “ALTER PROCEDURE statement” on page 456
- “CALL statement” on page 530
- “CREATE FUNCTION statement” on page 594
- “CREATE FUNCTION statement [Web service]” on page 586
- “CREATE PROCEDURE statement” on page 639
- “CREATE PROCEDURE statement [T-SQL]” on page 637
- “CREATE PROCEDURE statement [External call]” on page 620
- “DROP PROCEDURE statement” on page 765
- “SOAP structured data types” [SQL Anywhere Server - Programming]
- “HTTP and SOAP request structures” [SQL Anywhere Server - Programming]
- “HTTP request header management” [SQL Anywhere Server - Programming]
- “Tutorial: Using SQL Anywhere to access a SOAP/DISH service” [SQL Anywhere Server - Programming]
- “Tutorial: Create a web server and access it from a web client” [SQL Anywhere Server - Programming]
- “Variables supplied to web services” [SQL Anywhere Server - Programming]
- “GRANT statement” on page 827
- “SQL Anywhere as an HTTP web server” [SQL Anywhere Server - Programming]
- “Web client application development” [SQL Anywhere Server - Programming]
- “remote_idle_timeout option” [SQL Anywhere Server - Database Administration]
- “Substitution parameters used for clause values” [SQL Anywhere Server - Programming]
- “Web client application development” [SQL Anywhere Server - Programming]
- “ClientPort (CPORT) protocol option (client side only)” [SQL Anywhere Server - Database Administration]
- “HTTP request failed. Status code ‘%1’” [Error Messages]

Standards and compatibility

- SQL/2008 Vendor extension.
- Transact-SQL Not supported by Adaptive Server Enterprise.

Example

The following example creates a web service client procedure named FtoC.

```sql
CREATE PROCEDURE FtoC(
    IN temperature FLOAT,
    INOUT inoutheader LONG VARCHAR,
    IN inheader LONG VARCHAR
)
URL 'http://localhost:8082/FtoCService'
TYPE 'SOAP:DOC'
SOAPHEADER '!!inoutheader!inheader';
```

The following example creates a secure web service client procedure named SecureSendWithMimeType that uses a certificate stored in the database.

```sql
CREATE CERTIFICATE client_cert
FROM FILE 'C:\Users\Public\Documents\SQL Anywhere 16\Samples\Certificates\rsaroot.crt';

CREATE PROCEDURE SecureSendWithMimeType(
    value LONG VARCHAR,
);```
CALL SecureSendWithMimeType('<hello>this is xml</hello>', 'text/xml', 'https://localhost:4043/EchoService');

CREATE PROCEDURE statement [T-SQL]

Creates a new procedure in the database in a manner compatible with Adaptive Server Enterprise.

Syntax

The following subset of the Transact-SQL CREATE PROCEDURE statement is supported in SQL Anywhere.

```
CREATE [ OR REPLACE ] PROCEDURE [owner.]procedure_name
[ NO RESULT SET ]
[ (([ @parameter-name ] data-type [ = default ] [ OUTPUT ], ... [ ]) ]
[ WITH RECOMPILE ] AS statement-list
```

Parameters

- **OR REPLACE clause**  Specifying OR REPLACE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

- **NO RESULT SET clause**  Declares that no result set is returned by this procedure. This is useful when an external environment needs to know that a procedure does not return a result set.

- **WITH RECOMPILE clause**  This clause is accepted for Transact-SQL compatibility, but is ignored. SQL Anywhere always recompiles procedures the first time they are executed after a database is started, and stores the compiled procedure until the database is stopped.

Remarks

The following differences between Transact-SQL and SQL Anywhere statements (Watcom SQL) are listed to help those writing in both dialects.

- **Variable names prefixed by @**  The @ sign denotes a Transact-SQL variable name, while Watcom SQL variables can be any valid identifier, and the @ prefix is optional.

- **Input and output parameters**  Watcom SQL procedure parameters are INOUT by default or can be specified as IN, OUT, or INOUT. Transact-SQL procedure parameters are INPUT parameters by default. They can be specified as input/output with the addition of the OUTPUT keyword. There are no output-only parameters in the Transact-SQL dialect.

When you use the Watcom SQL dialect to declare a parameter OUT, it is output-only. The mixing of dialects is not recommended because it can cause problems when the procedure declaration is
unloaded and used to rebuild the database. If the procedure declaration is unloaded and used to rebuild the database, the rebuilt procedure declaration is in the Transact-SQL dialect, the OUTPUT keyword is used, and the parameter is input/output.

- **Parameter default values**  Watcom SQL procedure parameters are given a default value using the keyword DEFAULT, while Transact-SQL uses an equality sign (=) to provide the default value.

- **Returning result sets**  Watcom SQL uses a RESULT clause to specify returned result sets. In Transact-SQL procedures, the column names or alias names of the first query are returned to the calling environment.

The following Transact-SQL procedure illustrates how result sets are returned from Transact-SQL stored procedures:

```sql
CREATE PROCEDURE showdept @deptname varchar(30)  
AS  
    SELECT Employees.Surname, Employees.GivenName  
    FROM Departments, Employees  
    WHERE Departments.DepartmentName = @deptname  
    AND Departments.DepartmentID = Employees.DepartmentID;
```

The following is the corresponding Watcom SQL procedure:

```sql
CREATE PROCEDURE showdept2(in deptname varchar(30) )  
RESULT ( lastname char(20), firstname char(20) )  
ON EXCEPTION RESUME  
BEGIN  
    SELECT Employees.Surname, Employees.GivenName  
    FROM Departments, Employees  
    WHERE Departments.DepartmentName = deptname  
    AND Departments.DepartmentID = Employees.DepartmentID  
END;
```

- **Procedure body**  The body of a Transact-SQL procedure is a list of Transact-SQL statements prefixed by the AS keyword. The body of a Watcom SQL procedure is a compound statement, bracketed by BEGIN and END keywords.

**Privileges**

You must have the CREATE PROCEDURE privilege to create procedures owned by you. You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT privilege to create procedures owned by others.

**Side effects**

Automatic commit.

**See also**

- “CREATE FUNCTION statement” on page 594  
- “CREATE PROCEDURE statement” on page 639

**Standards and compatibility**

- **SQL/2008**  Transact-SQL extension.
• **Transact-SQL** SQL Anywhere supports a subset of the Adaptive Server Enterprise CREATE PROCEDURE statement syntax.

Only Transact-SQL SQL procedures are supported in the SQL Anywhere Transact-SQL dialect. To create an external procedure you must use Watcom SQL syntax. Adaptive Server Enterprise does not support the NO RESULT SET clause. If the Transact-SQL WITH RECOMPILE optional clause is supplied, it is ignored. SQL Anywhere always recompiles procedures the first time they are executed after a database is started, and stores the compiled procedure until the database is stopped.

Groups of Transact-SQL procedures are not supported in SQL Anywhere.

### CREATE PROCEDURE statement

Creates a user-defined SQL procedure in the database.

**Syntax**

```sql
CREATE [ OR REPLACE | TEMPORARY ] PROCEDURE [ owner.]procedure-name
( [ parameter, ... ] )
[ SQL SECURITY { INVOKER | DEFINER } ]
[ RESULT ( result-column, ... ) | NO RESULT SET ]
[ ON EXCEPTION RESUME ]
compound-statement | AT location-string
```

- **parameter**: parameter-mode parameter-name data-type [ DEFAULT expression ]
- **SQLCODE**
- **SQLSTATE**

- **parameter-mode**: IN | OUT | INOUT

- **result-column**: column-name data-type

**Parameters**

You can create permanent stored procedures that call external or native procedures written in a variety of programming languages. You can use PROC as a synonym for PROCEDURE.

**OR REPLACE clause** Specifying OR REPLACE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. An error is returned if you attempt to replace a procedure that is already in use.

**TEMPORARY clause** Specifying CREATE TEMPORARY PROCEDURE means that the stored procedure is visible only by the connection that created it, and that it is automatically dropped when the connection is dropped. Temporary stored procedures can also be explicitly dropped. You cannot perform ALTER, GRANT, or REVOKE on them, and, unlike other stored procedures, temporary stored procedures are not recorded in the catalog or transaction log.

Temporary procedures execute with the privileges of their creator (current user), or specified owner. You can specify an owner for a temporary procedure when:
the temporary procedure is created within a permanent stored procedure

the owner of the temporary and permanent procedure is the same

To drop the owner of a temporary procedure, you must drop the temporary procedure first.

Temporary stored procedures can be created and dropped when connected to a read-only database, and they cannot be external procedures.

For example, the following temporary procedure drops the fictitious table called CustRank. For this example, the procedure assumes that the table name is unique and can be referenced by the procedure creator without specifying the table owner:

```sql
CREATE TEMPORARY PROCEDURE drop_table( IN @TableName char(128) )
BEGIN
  IF EXISTS ( SELECT * FROM SYS.SYSTAB WHERE table_name = @TableName ) THEN
    EXECUTE IMMEDIATE 'DROP TABLE "' || @TableName || '"';
    MESSAGE 'Table "' || @TableName || '" dropped' to client;
  END IF;
END;
CALL drop_table( 'CustRank' );
```

**Parameters** Parameter names must conform to the rules for other database identifiers such as column names. They must be a valid SQL data type.

Parameters can be prefixed with one of the keywords IN, OUT, or INOUT. If you do not specify one of these values, parameters are INOUT by default. The keywords have the following meanings:

- **IN** The parameter is an expression that provides a value to the procedure.
- **OUT** The parameter is a variable that could be given a value by the procedure.
- **INOUT** The parameter is a variable that provides a value to the procedure, and could be given a new value by the procedure.

When procedures are executed using the CALL statement, not all parameters need to be specified. If a default value is provided in the CREATE PROCEDURE statement, missing parameters are assigned the default values. If an argument is not provided in the CALL statement, and no default is set, an error is given.

SQLSTATE and SQLCODE are special OUT parameters that output the SQLSTATE or SQLCODE value when the procedure ends. The SQLSTATE and SQLCODE special values can be checked immediately after a procedure call to test the return status of the procedure.

The SQLSTATE and SQLCODE special values are modified by the next SQL statement. Providing SQLSTATE or SQLCODE as procedure arguments allows the return code to be stored in a variable.

Specifying CREATE OR REPLACE PROCEDURE creates a new procedure, or replaces an existing procedure with the same name. This clause changes the definition of the procedure, but preserves existing privileges. You cannot use the OR REPLACE clause with temporary procedures. An error is returned if the procedure being replaced is already in use. Open cursors for a connection are closed when a CREATE OR REPLACE PROCEDURE statement is executed.
RESULT clause  The RESULT clause declares the number and type of columns in the result set. The parenthesized list following the RESULT keyword defines the result column names and types. This information is returned by the embedded SQL DESCRIBE or by ODBC SQLDescribeCol when a CALL statement is being described.

Some procedures can produce more than one result set, with different numbers of columns, depending on how they are executed. For example, the following procedure returns two columns under some circumstances, and one in others.

```
CREATE PROCEDURE names( IN formal char(1))
BEGIN
  IF formal = 'n' THEN
    SELECT GivenName
    FROM GROUPO.Employees
  ELSE
    SELECT Surname, GivenName
    FROM GROUPO.Employees
  END IF
END;
```

Procedures with variable result sets must be written without a RESULT clause, or in Transact-SQL. Their use is subject to the following limitations:

- **Embedded SQL**  You must DESCRIBE the procedure call after the cursor for the result set is opened, but before any rows are returned, to get the proper shape of result set. The CURSOR `cursor-name` clause on the DESCRIBE statement is required.

- **ODBC, OLE DB, ADO.NET**  Variable result-set procedures can be used by applications using these interfaces. The proper description of the result sets is carried out by the driver or provider.

- **Open Client applications**  Variable result-set procedures can be used by Open Client applications.

- **Web services**  Web services rely on the RESULTS clause of the stored procedure to determine the number and types of the column in the result set. Web services do not support procedures that return multiple result sets, nor do they support variable result sets through the use of EXECUTE IMMEDIATE.

**Note**
If an EXECUTE IMMEDIATE statement that includes a WITH RESULT SET ON clause is used in the procedure, and if the result set that is returned from the statement is the same as the result set that is returned from the procedure, then only the first column of the EXECUTE IMMEDIATE statement’s result set is returned.

If your procedure returns only one result set, you should use a RESULT clause. The presence of this clause prevents ODBC and Open Client applications from re-describing the result set after a cursor is open.

To handle multiple result sets, ODBC must describe the currently executing cursor, not the procedure’s defined result set. Therefore, ODBC does not always describe column names as defined in the RESULT clause of the procedure definition. To avoid this problem, use column aliases in the SELECT statement that generates the result set.
**NO RESULT SET clause**  Declares that no result set is returned by this procedure. This is useful when an external environment needs to know that a procedure does not return a result set.

**SQL SECURITY clause**  The SQL SECURITY clause defines whether the procedure is executed as the INVOKER (the user who is calling the procedure), or as the DEFINER (the user who owns the procedure). The default is DEFINER.

When SQL SECURITY INVOKER is specified, more memory is used because annotation must be done for each user that calls the procedure. When SQL SECURITY INVOKER is specified, name resolution is done as the invoker as well. Therefore, make sure to qualify all object names (tables, procedures, and so on) with their appropriate owner. For example, suppose user1 creates the following procedure:

```sql
CREATE PROCEDURE user1.myProcedure()
    RESULT( columnA INT )
    SQL SECURITY INVOKER
BEGIN
    SELECT columnA FROM table1;
END;
```

If user2 attempts to run this procedure and a table user2.table1 does not exist, a table lookup error results. Additionally, if a user2.table1 does exist, that table is used instead of the intended user1.table1. To prevent this situation, qualify the table reference in the statement (user1.table1, instead of just table1).

**ON EXCEPTION RESUME clause**  This clause enables Transact-SQL-like error handling to be used within a Watcom SQL syntax procedure.

If you use ON EXCEPTION RESUME, the procedure takes an action that depends on the setting of the on_tsql_error option. If on_tsql_error is set to Conditional (the default) the execution continues if the next statement handles the error; otherwise, it exits.

Error-handling statements include the following:

- IF
- SELECT @variable =
- CASE
- LOOP
- LEAVE
- CONTINUE
- CALL
- EXECUTE
- SIGNAL
- RESIGNAL
- DECLARE
- SET VARIABLE

You should not use explicit error handling code with an ON EXCEPTION RESUME clause.

This clause is ignored within the TRY block of a BEGIN...END statement.

**compound-statement**  A set of SQL statements bracketed by BEGIN and END, and separated by semicolons.
**AT clause** Create a proxy stored procedure on the current database for a remote procedure specified by `location-string`. The AT clause supports the semicolon (;) as a field delimiter in `location-string`. If no semicolon is present, a period is the field delimiter. This allows file names and extensions to be used in the database and owner fields.

The string in the AT clause can also contain local or global variable names enclosed in braces (`{variable-name}`). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains 'bostonase.master.dbo.{@myprocedure}' indicates that `@myprocedure` is a SQL variable and that the current contents of the `@myprocedure` variable should be substituted when the remote procedure is used.

If a remote procedure can return a result set, even if it does not always return one, then the local procedure definition must contain a RESULT clause.

**Remarks**

The CREATE PROCEDURE statement creates a procedure in the database. A procedure is invoked with a CALL statement.

If a stored procedure returns a result set, it cannot set output parameters or return a return value.

When referencing a temporary table from multiple procedures, a potential issue can arise if the temporary table definitions are inconsistent and statements referencing the table are cached.

**Privileges**

You must have the CREATE PROCEDURE system privilege to create procedures owned by you. You must have the CREATE ANY PROCEDURE or CREATE ANY OBJECT privilege to create procedures owned by others. To create external procedures, you must also have the CREATE EXTERNAL REFERENCE system privilege.

You do not need any privilege to create temporary procedures.

**Side effects**

Automatic commit, even for temporary procedures.
Standards and compatibility

- **SQL/2008**  CREATE PROCEDURE is a core feature of the SQL/2008 standard, but some of its components supported in SQL Anywhere are optional SQL language features. A subset of these features includes:
  - The SQL SECURITY clause is optional SQL/2008 language feature T324.
  - The ability to pass a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY value to a SQL procedure is SQL/2008 language feature T041.
  - The ability to create or modify a schema object within a SQL procedure, using statements such as CREATE TABLE or DROP TRIGGER, is SQL/2008 language feature T651.
  - The ability to use a dynamic-SQL statement within a SQL procedure, including statements such as EXECUTE IMMEDIATE, PREPARE, and DESCRIBE, is SQL/2008 language feature T652.

Several clauses of the CREATE PROCEDURE statement are vendor extensions. These include:
  - The TEMPORARY clause.
  - The ON EXCEPTION RESUME clause.
  - The AT clause.
  - The optional DEFAULT clause for a specific routine parameter.
The RESULT and NO RESULT SET clauses. The SQL/2008 standard uses the RETURNS keyword.

The optional OR REPLACE clause.

- **Transact-SQL** CREATE PROCEDURE is supported by Adaptive Server Enterprise.

**Examples**

The following procedure queries the Employees table and returns salaries that are within the specified percent (*percentage*) of a specified salary (*sal*):

```sql
CREATE OR REPLACE PROCEDURE AverageEmployees( IN percentage NUMERIC( 5,3),
   IN sal NUMERIC( 20, 3 ) )
RESULT( Department CHAR(40), GivenName person_name_t, Surname person_name_t,
   Salary NUMERIC( 20, 3) )
BEGIN
   DECLARE maxS NUMERIC( 20, 3 );
   DECLARE minS NUMERIC( 20, 3 );

   IF percentage >= 1 THEN
      SET percentage = percentage / 100;
   ELSEIF percentage < 0 THEN
      SELECT 'Percentage error', 'Err','Err', -1;
      RETURN;
   END IF;

   SELECT MIN( E.Salary ), MAX( E.Salary ) INTO minS, maxS
   FROM GROUPO.Employees E;

   IF sal < minS OR sal > maxS THEN
      SELECT 'Salary out of bounds', 'Err', 'Err', -2;
      RETURN;
   END IF;

   SELECT D.DepartmentName, E.GivenName, E.Surname, E.Salary
   FROM GROUPO.Employees E JOIN Departments D ON E.DepartmentID =
   D.DepartmentID
   WHERE E.Salary BETWEEN sal *( 1 - percentage ) AND sal * ( 1 +
   percentage );
END;
```

The following procedure uses a CASE statement to classify the results of a query.

```sql
CREATE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
   DECLARE prod_name CHAR(20);
   SELECT name INTO prod_name FROM GROUPO.Products
   WHERE ID = product_ID;
   CASE prod_name
      WHEN 'Tee Shirt' THEN
         SET type = 'Shirt'
      WHEN 'Sweatshirt' THEN
         SET type = 'Shirt'
      WHEN 'Baseball Cap' THEN
         SET type = 'Hat'
      WHEN 'Visor' THEN
         SET type = 'Hat'
      WHEN 'Shorts' THEN
         SET type = 'Shorts'
      ELSE
         SQL statements
   END;
```
The following example replaces the ProductType procedure created in the previous example. After replacing the procedure, the parameters for Tee Shirt and Sweatshirt are updated:

```sql
CREATE OR REPLACE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
    DECLARE prod_name CHAR(20);
    SELECT name INTO prod_name FROM GROUPO.Products
    WHERE ID = product_ID;
    CASE prod_name
    WHEN 'Tee Shirt' THEN
        SET type = 'T Shirt'
    WHEN 'Sweatshirt' THEN
        SET type = 'Long Sleeve Shirt'
    WHEN 'Baseball Cap' THEN
        SET type = 'Hat'
    WHEN 'Visor' THEN
        SET type = 'Hat'
    WHEN 'Shorts' THEN
        SET type = 'Shorts'
    ELSE
        SET type = 'UNKNOWN'
    END CASE;
END;
```

The following procedure uses a cursor and loops over the rows of the cursor to return a single value.

```sql
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
    DECLARE err_notfound EXCEPTION
    FOR SQLSTATE '02000';
    DECLARE curThisCust CURSOR FOR
        SELECT CompanyName,
        CAST(SUM(SalesOrderItems.Quantity * 
        Products.UnitPrice) AS INTEGER) VALUE
        FROM GROUPO.Customers
        LEFT OUTER JOIN SalesOrders
        LEFT OUTER JOIN SalesOrderItems
        LEFT OUTER JOIN Products
        GROUP BY CompanyName;
    DECLARE ThisValue INT;
    DECLARE ThisCompany CHAR(35);
    SET TopValue = 0;
    OPEN curThisCust;
    CustomerLoop:
    LOOP
        FETCH NEXT curThisCust
        INTO ThisCompany, ThisValue;
        IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop;
        END IF;
        IF ThisValue > TopValue THEN
            SET TopValue = ThisValue;
            SET TopCompany = ThisCompany;
        END IF;
    END LOOP CustomerLoop;
    CLOSE curThisCust;
END;
```
The following example creates the procedure `NewDepartment`, which performs an `INSERT` into the `Departments` table of the SQL Anywhere sample database, creating a new department.

```sql
CREATE PROCEDURE NewDepartment(
    IN id INT,
    IN name CHAR(35),
    IN head_id INT )
BEGIN
    INSERT INTO GROUPO.Departments ( DepartmentID, DepartmentName, DepartmentHeadID )
    VALUES ( id, name, head_id );
END;
```

The body of a procedure is a compound statement. The compound statement starts with a `BEGIN` statement and concludes with an `END` statement. For `NewDepartment` the compound statement is a single `INSERT` bracketed by `BEGIN` and `END` statements.

Parameters to procedures can be marked as one of IN, OUT, or INOUT. By default, parameters are INOUT parameters. All parameters to the `NewDepartment` procedure are IN parameters, as they are not changed by the procedure. You should set parameters to IN if they are not used to return values to the caller.

### CREATE PUBLICATION statement [MobiLink] [SQL Remote]

Creates a publication. In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

#### Syntax 1 (MobiLink general use)

```sql
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner. ] publication-name
    ( article-definition, ... )

article-definition :
    TABLE table-name [ ( column-name, ... ) ]
    [ WHERE search-condition ]
```

#### Syntax 2 (MobiLink scripted upload)

```sql
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner. ] publication-name
    WITH SCRIPTED UPLOAD
    ( article-definition, ... )

article-definition :
    TABLE table-name [ ( column-name, ... ) ]
    [ USING ( [ PROCEDURE ] [ owner. ] procedure-name
    FOR UPLOAD { INSERT | DELETE | UPDATE }, ... ) ]
```

#### Syntax 3 (MobiLink download-only publications)

```sql
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner. ] publication-name
    FOR DOWNLOAD ONLY
    ( article-definition, ... )
```
article-definition : TABLE table-name [ ( column-name, ... ) ]

Syntax 4 (SQL Remote)
CREATE PUBLICATION [ IF NOT EXISTS ] [ owner.] publication-name
( article-definition, ... )

article-definition :
   TABLE table-name [ ( column-name, ... ) ]
   [ WHERE search-condition ]
   [ SUBSCRIBE BY expression ]

Parameters
IF NOT EXISTS clause When the IF NOT EXISTS clause is specified and the named publication already exists, no changes are made and an error is not returned.

article-definition Publications are built from articles. Each article identifies the rows and columns of a single table that are included in the publication. A publication may not contain two articles that refer to the same table.

If a list of column-names is included in an article, only those columns are included in the publication. If no column-names are listed, all columns in the table are include in the publication. For MobiLink synchronization, if column-names are listed then all columns in the primary key of the table must be included in the list.

In Syntax 2, which is used for publications that perform scripted uploads, the article description also registers the scripts that are used to define the upload.

In Syntax 3, which is used for download-only publications, the article specifies only the tables and columns to be downloaded.

WHERE clause The WHERE clause lets you define the subset of rows in a table to be included in an article.

In MobiLink applications, the WHERE clause affects the rows included in the upload. (The download is defined by the download_cursor script.) In MobiLink SQL Anywhere remote databases, the WHERE clause can only refer to columns included in the article, and cannot contain subqueries, variables, or non-deterministic functions.

SUBSCRIBE BY clause In SQL Remote, one way of defining a subset of rows of a table to be included in an article is to use a SUBSCRIBE BY clause. This clause allows many different subscribers to receive different rows from a table in a single publication definition.

Remarks
The CREATE PUBLICATION statement creates a publication in the database. A publication can be created for another user by specifying an owner name.

In MobiLink, publications are required in SQL Anywhere remote databases, and are optional in UltraLite databases. These publications and the subscriptions to them determine which data is uploaded to the MobiLink server.
You set options for a MobiLink publication with the ADD OPTION clause in the CREATE SYNCHRONIZATION SUBSCRIPTION statement or ALTER SYNCHRONIZATION SUBSCRIPTION statement.

Syntax 2 creates a publication for scripted uploads. Use the USING clause to register the stored procedures that you want to use to define the upload. For each table, you can use up to three stored procedures: one each for inserts, deletes, and updates.

Syntax 3 creates a download-only publication that can be synchronized with no transaction log file. When download-only publications are synchronized, downloaded rows may overwrite changes that were made to those rows in the remote database.

In SQL Remote, publishing is a two-way operation, as data can be entered at both consolidated and remote databases. In a SQL Remote installation, any consolidated database and all remote databases must have the same publication defined. Running the SQL Remote Extraction utility from a consolidated database automatically executes the correct CREATE PUBLICATION statement in the remote database.

For all syntaxes, you must have exclusive access to all tables referred to in the statement to execute the statement.

**Privileges**

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

**Side effects**

Automatic commit.

**See also**

- “ALTER PUBLICATION statement [MobiLink] [SQL Remote]” on page 457
- “DROP PUBLICATION statement [MobiLink] [SQL Remote]” on page 766
- “CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” on page 686
- “ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” on page 482
- SQL Anywhere MobiLink clients: “Publications” [MobiLink - Client Administration]
- UltraLite MobiLink clients: “CREATE PUBLICATION statement [UltraLite]” [UltraLite - Database Management and Reference]
- SQL Remote: “Publications and articles” [SQL Remote]
- “Scripted upload” [MobiLink - Client Administration]
- “Download-only publications” [MobiLink - Client Administration]
- “SYSSYNC system view” on page 1396
- “Publications for scripted upload” [MobiLink - Client Administration]
- “Creating publications” [SQL Remote]
- “Publishing only some columns in a table” [SQL Remote]
- “Publishing only some rows using the SUBSCRIBE BY clause” [SQL Remote]
- “Publishing only some rows using a WHERE clause” [SQL Remote]
- “Publish only some rows in a table” [SQL Remote]
- “Replicating the primary key pool” [SQL Remote]
- “Overlap partitions” [SQL Remote]
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement publishes all columns and rows of two tables.

```
CREATE PUBLICATION pub_contact (
    TABLE GROUPO.Contacts,
    TABLE GROUPO.Customers
);
```

The following statement publishes only some columns of one table.

```
CREATE PUBLICATION pub_customer ( 
    TABLE GROUPO.Customers ( ID, CompanyName, City )
);
```

The following statement publishes only the rows for customer located in New York (NY) by including a WHERE clause that tests the State column of the Customers table.

```
CREATE PUBLICATION pub_customer ( 
    TABLE GROUPO.Customers ( ID, CompanyName, City, State, Status )
WHERE State = 'NY'
);
```

The following statement publishes only some rows by providing a subscribe-by value. This method can be used only with SQL Remote.

```
CREATE PUBLICATION pub_customer ( 
    TABLE GROUPO.Customers ( ID, CompanyName, City, State )
    SUBSCRIBE BY State
);
```

The subscribe-by value is used as follows when you create a SQL Remote subscription.

```
CREATE SUBSCRIPTION TO pub_customer ( 'NY' ) 
    FOR jsmith;
```

The following example creates a MobiLink publication that uses scripted uploads:

```
CREATE PUBLICATION pub WITH SCRIPTED UPLOAD ( 
    GROUPO.TABLE t1 (a, b, c) USING (  
        PROCEDURE my.t1_ui FOR UPLOAD INSERT,  
        PROCEDURE my.t1_ud FOR UPLOAD DELETE,  
        PROCEDURE my.t1_uu FOR UPLOAD UPDATE  
    ),  
    GROUPO.TABLE t2 AS my_t2 USING (  
        PROCEDURE my.t2_ui FOR UPLOAD INSERT  
    )
);
```

The following example creates a download-only publication:

```
CREATE PUBLICATION p1 FOR DOWNLOAD ONLY (  
    GROUPO.TABLE t1  
);
```
CREATE REMOTE [MESSAGE] TYPE statement [SQL Remote]

Identifies a message link and return address for outgoing messages from a database.

Syntax

```
CREATE REMOTE [MESSAGE] TYPE message-system
  [ ADDRESS address-string ]
```

- `message-system`: One of the message systems supported by SQL Remote. It must be one of the following values: FILE, FTP, or SMTP.
- `address-string`: A string containing a valid address for the specified message system.

Parameters

- `message-system`: One of the message systems supported by SQL Remote. It must be one of the following values: FILE, FTP, or SMTP.
- `address-string`: A string containing a valid address for the specified message system.

Remarks

The Message Agent sends outgoing messages from a database using one of the supported message links. Return messages for users employing the specified link are sent to the specified address as long as the remote database is created by the Extraction utility. The Message Agent starts links only if it has remote users for those links.

The address is the publisher's address under the specified message system. If it is an email system, the address string must be a valid email address. If it is a file-sharing system, the address string is a subdirectory of the directory set in the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.

To remove the address, execute a CREATE REMOTE MESSAGE TYPE statement without an ADDRESS clause.

The dbinit utility creates message types automatically, without an address. Unlike other CREATE statements, the CREATE REMOTE MESSAGE TYPE statement does not give an error if the type exists; instead it alters the type.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.
Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

When remote databases are extracted using the extraction utility, the following statement sets all recipients of file message-system messages to send messages back to the *company* subdirectory.

The statement also instructs dbremote to look in the *company* subdirectory for incoming messages.

```
CREATE REMOTE MESSAGE TYPE file
ADDRESS 'company';
```

**CREATE ROLE statement**

Creates or replaces a role, creates a user-extended role, or modifies administrators for a role.

**Syntax**

```
CREATE [ OR REPLACE ] ROLE { role-name | FOR USER userid }
    [ WITH ADMIN [ ONLY ] administrator-userid [,...] ]
```

**Parameters**

- **role-name**  Use this variable to assign a name to the new role. This name must be unique across all users and roles in the database.

- **OR REPLACE clause**  Use this clause to create the role if it does not already exist or replace its administrators if it does exist.

- **FOR USER userid clause**  Use this clause to convert the specified user into a user-extended role that can be assigned to others. The user must not already be extended as another role.

- **WITH ADMIN and WITH ADMIN ONLY administrator-userid clause**  Optionally specify administrators for the role. WITH ADMIN means that *administrator-userid* can exercise the role and administer it. WITH ADMIN ONLY means that *administrator-userid* can only administer the role. If no clause is specified, then any user with the MANAGE ROLES system privilege can administer the role.
The min_role_admins database option controls the minimum number of administrators required for each role. If you do not specify enough administrators when creating the role, the statement returns an error.

Remarks
The name of the new role must not begin and end with 'SYS_' and '_ROLE', respectively. For example SYS_MyBackup_ROLE is not an acceptable name for a user-defined role, whereas MyBackup_ROLE and SYS_MyBackup are acceptable.

If an ADMIN clause is specified, only the specified users can administer the roles. If no ADMIN clause is specified, then by default the role is granted to the MANAGE ROLES system privilege, with administrative rights only. This means that global administrators can administer the role.

To create a user-extended role (that is, to extend a user to be a role), use the CREATE ROLE FOR USER userid syntax.

Use the GRANT statements to grant system privileges to the role.

Privileges
You must have the MANAGE ROLES system privilege to create a new role.

If the OR REPLACE clause is specified and the role already exists, you must also have administrative rights over the role.

Side effects
None.

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement creates the Sales role. Any user with the MANAGE ROLES system privilege can administer the role.

CREATE ROLE Sales;

The following statement extends user JaneSmith to become a role that can be assigned to others.

CREATE ROLE FOR USER JaneSmith;

The following statement creates the role Finance with MaryJones and JeffTurkott as role administrators with administrative rights (only) for the role.

CREATE ROLE Finance
WITH ADMIN ONLY MaryJones, JeffTurkott;

The following example replaces the existing Finance role created in the previous example, replacing MaryJones and JeffTurkott with EllenChong and DaveLexx as role administrators, this time with exercise rights to the role.
CREATE OR REPLACE ROLE Finance
WITH ADMIN EllenChong, DaveLexx;

See also

- “Role administrators” [SQL Anywhere Server - Database Administration]
- “min_role_admins option” [SQL Anywhere Server - Database Administration]
- “DROP ROLE statement” on page 769
- “Granting a role (Sybase Central)” [SQL Anywhere Server - Database Administration]
- “User-extended roles” [SQL Anywhere Server - Database Administration]

CREATE SCHEMA statement

Creates a collection of tables and views for a database user.

Syntax

```
CREATE SCHEMA
AUTHORIZATION userid
[ create-table-statement
  | create-view-statement
  | grant-statement
] ...
```

Remarks

The CREATE SCHEMA statement creates a schema. A schema is a collection of tables and views along with their associated privileges.

The `userid` must be the user ID of the current connection. You cannot create a schema for another user.

If any statement contained in the CREATE SCHEMA statement fails, the entire CREATE SCHEMA statement is rolled back.

The CREATE SCHEMA statement is a way of collecting together individual CREATE and GRANT statements into one operation. There is no SCHEMA database object created in the database, and to drop the objects you must use individual DROP TABLE or DROP VIEW statements. To revoke privileges, you must use a REVOKE statement for each privilege granted.

The individual CREATE or GRANT statements are not separated by statement delimiters. The statement delimiter marks the end of the CREATE SCHEMA statement itself.

The individual CREATE or GRANT statements must be ordered such that the objects are created before privileges are granted on them.

Although you can create more than one schema for a user, doing so is not recommended.

Privileges

The system privileges required depend on the operation specified in the CREATE SCHEMA statement you define. For information about the required system privileges required, see the System privileges sections for applicable statements (CREATE TABLE, CREATE VIEW, and GRANT).
Side effects

Automatic commit.

See also

- “CREATE TABLE statement” on page 690
- “CREATE VIEW statement” on page 724
- “GRANT statement” on page 827

Standards and compatibility

- **SQL/2008**  CREATE SCHEMA is a core feature of the SQL/2008 standard. The ability to create multiple schemas for a single user is SQL/2008 optional language feature F171. SQL Anywhere does not support the use of REVOKE statements within the CREATE SCHEMA statement, and does not allow their use within Transact-SQL batches or procedures.

- **Transact-SQL**  Supported by Adaptive Server Enterprise, which supports GRANT and REVOKE statements within the CREATE SCHEMA statement.

Example

The following CREATE SCHEMA statement creates a schema consisting of two tables. The statement must be executed by the user ID sample_user, who must have the CREATE TABLE system privilege. If the statement creating table t2 fails, neither table is created.

```sql
CREATE SCHEMA AUTHORIZATION sample_user
CREATE TABLE t1 ( id1 INT PRIMARY KEY )
CREATE TABLE t2 ( id2 INT PRIMARY KEY );
```

The statement delimiter in the following CREATE SCHEMA statement is placed after the first CREATE TABLE statement. As the statement delimiter marks the end of the CREATE SCHEMA statement, the example is interpreted as a two statement batch by the database server. If the statement creating table t2 fails, the table t1 is still created.

```sql
CREATE SCHEMA AUTHORIZATION sample_user
CREATE TABLE t1 ( id1 INT PRIMARY KEY );
CREATE TABLE t2 ( id2 INT PRIMARY KEY );
```

**CREATE SEQUENCE statement**

Creates a sequence that can be used to generate primary key values that are unique across multiple tables, and for generating default values for a table.

**Syntax**

```sql
CREATE [ OR REPLACE ]
SEQUENCE [ owner. ] sequence-name
[ INCREMENT BY signed-integer ]
[ START WITH signed-integer ]
[ MINVALUE signed-integer | NO MINVALUE ]
[ MAXVALUE signed-integer | NO MAXVALUE ]
[ CACHE integer | NO CACHE ]
[ CYCLE | NO CYCLE ]
```
Parameters

**OR REPLACE clause** Specifying OR REPLACE creates a new sequence, or replaces an existing sequence with the same name. If you do not use the OR REPLACE clause, an error is returned if you specify the name of a sequence that already exists for the current user.

**INCREMENT BY clause** Defines the amount the next sequence value is incremented from the last value assigned. The default is 1. Specify a negative value to generate a descending sequence. An error is returned if the INCREMENT BY value is 0.

**START WITH clause** Defines the starting sequence value. If you do not specify a value for the START WITH clause, MINVALUE is used for ascending sequences and MAXVALUE is used for descending sequences. An error is returned if the START WITH value is beyond the range specified by MINVALUE or MAXVALUE.

**MINVALUE clause** Defines the smallest value generated by the sequence. The default is 1. An error is returned if MINVALUE is greater than \(2^{63}-1\) or less than \(-2^{63}-1\). An error is also returned if MINVALUE is greater than MAXVALUE.

**MAXVALUE clause** Defines the largest value generated by the sequence. The default is \(2^{63}-1\). An error is returned if MAXVALUE is greater than \(2^{63}-1\) or less than \(-2^{63}-1\).

**CACHE clause** Specifies the number of preallocated sequence values that are kept in memory for faster access. When the cache is exhausted, the sequence cache is repopulated and a corresponding entry is written to the transaction log. At checkpoint time, the current value of the cache is forwarded to the ISYSSEQUENCE system table. The default is 100.

**CYCLE clause** Specifies whether values should continue to be generated after the maximum or minimum value is reached. The default is NO CYCLE, which returns an error once the maximum or minimum value is reached.

Remarks

A sequence is a database object that allows the automatic generation of numeric values. A sequence is not bound to a specific or unique table column and is only accessible through the table column to which it is applied.

Sequences can generate values in one of the following ways:

- Increment or decrement monotonically without bound
- Increment or decrement monotonically to a user-defined limit and stop
- Increment or decrement monotonically to a user-defined limit and cycle back to the beginning and start again

You control the behavior when the sequence runs out of values using the CYCLE clause.

If a sequence is increasing and it exceeds the MAXVALUE, MINVALUE is used as the next sequence value if CYCLE is specified. If a sequence is decreasing and it falls below MINVALUE, MAXVALUE is used as the next sequence value if CYCLE is specified. If CYCLE is not specified, an error is returned.

Sequence values cannot be used with views or materialized view definitions.
Privileges

You must have the CREATE ANY SEQUENCE or CREATE ANY OBJECT system privilege to create sequences.

Side effects

None

See also

- “ALTER SEQUENCE statement” on page 462
- “DROP SEQUENCE statement” on page 771
- “sequence-expression clause, SELECT statement” on page 963
- “Use of a sequence to generate unique values” [SQL Anywhere Server - SQL Usage]
- “Choosing between sequences and AUTOINCREMENT values” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008  Sequences comprise SQL/2008 language feature T176. SQL Anywhere does not allow optional specification of the sequence data type—this behavior can be achieved with a CAST when using the sequence.

  In addition, the following are vendor extensions:

  - CACHE clause
  - OR REPLACE syntax
  - CURRVAL expression
  - Use of sequences in DEFAULT expressions

Example

The following example creates a sequence named Test that starts at 4, increments by 2, does not cycle, and caches 15 values at a time:

```
CREATE SEQUENCE Test
START WITH 4
INCREMENT BY 2
NO MAXVALUE
NO CYCLE
CACHE 15;
```

CREATE SERVER statement

Creates a remote server.

Syntax 1

```
CREATE SERVER server-name
CLASS server-class-string
USING connection-info-string
[ READ ONLY ]
```
server-class-string :
'ADSODBC' | 'ASEODBC' | 'DB2ODBC' | 'HANAODBC' | 'IQODBC' | 'MIRROR' | 'MSACCESSODBC' | 'MSSODBC' | 'MYSQLODBC' | 'ODBC' | 'ORAODBC' | 'SAODBC' | 'ULODBC'

collection-info-string :
{ 'data-source-name' | 'sql anywhere-connection-string' }

Syntax 2
CREATE SERVER server-name CLASS 'DIRECTORY'
USING using-string

using-string :
'WOR = path [ ;SUBDIRS = n ] [ ;READONLY = { YES | NO } ] [ ;CREATEDIRS = { YES | NO } ]
[ ;DELIMITER = { / \ } ]'

Parameters
CLASS clause  Specifies the server class you want to use for a remote connection. Server classes contain detailed server capability information.

The DIRECTORY class is used in Syntax 2 to access a directory on the local computer.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

USING clause  In Syntax 1, the USING clause supplies a connection string for the database server. The appropriate connection string depends on the driver being used, which in turn depends on the server-class-string.

The USING clause is an ODBC connection string that can include 'DSN=data-source-name' to specify an ODBC data source name and/or 'DRIVER=driver-name' to specify a driver binary on Unix or driver name on Windows.

For SQL Anywhere remote servers (SAODBC server classes), the connection-info-string parameter can be any valid SQL Anywhere connection string. You can use any SQL Anywhere connection parameters. For example, if you have connection problems, you can include a LOG connection parameter to troubleshoot the connection attempt.

The string in the USING clause can also contain local or global variable names enclosed in braces ({variable-name}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, a USING clause that contains 'DSN={@mydsn}' indicates that @mydsn is a SQL variable.
and that the current contents of the @mydsn variable should be substituted when a connection is made to the remote data access server.

In Syntax 2, the USING clause specifies the following values for the local directory:

- **ROOT clause**  Specifies the path, relative to the database server, that is the root of the directory access class. When you create a proxy table using the directory access server name, the proxy table is relative to this root path.

  If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

- **SUBDIRS clause**  Specifies a number between 0 and 10 that represents the number of levels of directories within the root that the database server can access. If SUBDIRS is omitted or set to 0, then only the files in the root directory are accessible via the directory access server. You can create proxy tables to any of the directories or subdirectories available via the directory access server.

- **READONLY clause**  Specifies whether the files accessed by the directory are READONLY and cannot be modified. By default, READONLY is set to NO.

- **CREATEDIRS clause**  Specifies whether directories can be created using the directory access server. The default is NO.

- **DELIMITER clause**  Specifies whether paths in the file_name column are delimited by a slash (/) or backslash (\) character. By default, the native path delimiter is used.

The string in the USING clause can also contain local or global variable names enclosed in braces ({variable-name}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, a USING clause that contains '"ROOT=[@mypath]"' indicates that @mypath is a SQL variable and that the current contents of the @mypath variable should be substituted when a connection to the directory access server is established. This permits the creation of dynamic directory access servers.

For more information about using variables in the USING clause, see “Creating directory access servers (SQL)” [SQL Anywhere Server - SQL Usage].

**READ ONLY clause**  Specifies that the remote server is accessed in read-only mode.

**Remarks**

This statement is not supported on Windows Mobile.

When you create a remote server, it is added to the ISYSSERVER system table. Use the corresponding system view SYSSERVER to view the table.

- **Syntax 1**  The CREATE SERVER statement defines a remote server.

  To bypass the ODBC driver manager when defining a SQL Anywhere remote server, use the syntax below, followed by the remainder of the connection-info-string:
CREATE SERVER remote-server
CLASS 'SAOdbc'
USING 'DRIVER=SQL Anywhere Native;DSN=my-dsn;UID=my-username;PWD=my-pwd';

This syntax allows remote data access to load the SQL Anywhere ODBC driver directly and is supported by Windows and Unix. Loading the SQL Anywhere ODBC driver directly ensures that the ODBC driver for the current server version is used. Also, if the SQL Anywhere ODBC driver is only used for remote data access, it does not need to be registered.

Note
If the application also makes use of non-SQL Anywhere remote servers, or if there are SQL Anywhere remote servers defined without using 'DRIVER=SQL Anywhere Native', then remote data access still uses a driver manager for the other remote servers.

On Unix platforms you can also reference the SQL Anywhere ODBC driver. The syntax is as follows:

USING 'DRIVER=SQL Anywhere 16;DSN=my-dsn'

- Syntax 2 The CREATE SERVER statement lets you create a directory access server that accesses the local directory structure on the computer where the database server is running. You must create an external login for each database user that needs to use the directory access server. On Unix, the database server runs as a specific user, so file permissions are based on the privileges granted to the database server user.

Privileges
You must have the SERVER OPERATOR system privilege.

Side effects
Automatic commit.

See also
- “ALTER SERVER statement” on page 463
- “CREATE EXTERNLOGIN statement” on page 578
- “CREATE EXISTING TABLE statement” on page 576
- “DROP SERVER statement” on page 772
- “DROP REMOTE CONNECTION statement” on page 768
- “SYSSERVER system view” on page 1391
- “Remote data access” [SQL Anywhere Server - SQL Usage]
- “Remote servers” [SQL Anywhere Server - SQL Usage]
- “Directory access servers” [SQL Anywhere Server - SQL Usage]
- “Server classes for remote data access” [SQL Anywhere Server - SQL Usage]
- “Connection parameters” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.
Example

The following example creates a SQL Anywhere remote server named RemoteSA, using the SQL Anywhere ODBC driver.

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere 16;DSN=RemoteDS';
```

The following example directly loads the SQL Anywhere ODBC driver without using the ODBC driver manager:

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere Native;DSN=RemoteDS';
```

The following example uses a variable reference to create a dynamic remote data access server. You need the MANAGE ANY USER and CREATE TABLE system privileges to run this example.

```
CREATE SERVER RemoteSA
CLASS 'SAODBC'
USING 'DRIVER=SQL Anywhere
16;DSN={dsn_string};Server=saremote;UID=DBA;PWD=sql';

CREATE VARIABLE dsn_string LONG VARCHAR;
SET dsn_string = 'Test16';

CREATE EXTERNLOGIN DBA TO RemoteSA;

CREATE EXISTING TABLE test_employees
AT 'RemoteSA...Employees';
SELECT * FROM test_employees;
DROP REMOTE CONNECTION TO RemoteSA CLOSE ALL;
```

The following example creates an Adaptive Server Enterprise (ASE) remote server named ase_prod using the ASE ODBC driver.

```
CREATE SERVER ase_prod
CLASS 'ASEODBC'
USING 'DSN=remoteASE';
```

The following example creates a remote server for the Oracle server named oracle723. Its ODBC data source name is oracle723.

```
CREATE SERVER oracle723
CLASS 'ORAODBC'
USING 'oracle723';
```

The following example creates a directory access server that only sees files within the directory c:\temp.

```
CREATE SERVER diskserver0
CLASS 'DIRECTORY'
USING 'ROOT=c:\\temp';
CREATE EXTERNLOGIN DBA TO diskserver0;
CREATE EXISTING TABLE diskdir0 AT 'diskserver0;;;.';

-- Get a list of those files.
SELECT privileges, file_name, size FROM diskdir0;
```
The following example creates a dynamic directory access server that is used to explore two different directories.

```
CREATE SERVER diskserver9
CLASS 'DIRECTORY'
USING '{dir_options}';

CREATE EXTERNLOGIN DBA TO diskserver9;
CREATE EXISTING TABLE diskdir9 AT 'diskserver9;;;.';

CREATE VARIABLE dir_options VARCHAR(256);
SET dir_options = 'ROOT=c:\temp;SUBDIRS=9;DELIMITER=/';
SELECT * FROM diskdir9;

DROP REMOTE CONNECTION TO diskserver9 CLOSE ALL;
SET dir_options = 'ROOT=c:\ProgramData;SUBDIRS=9;DELIMITER=/';
SELECT * FROM diskdir9;
```

**CREATE SERVICE statement [HTTP web service]**

Creates a new HTTP web service.

**Syntax**

```
CREATE SERVICE service-name
TYPE {'RAW' | 'HTML' | 'JSON' | 'XML'}
[ URL [ PATH [ ON | OFF | ELEMENTS ] ]
[ common-attributes ]
[ AS { statement | NULL } ]
```

**common-attributes**:  
- `AUTHORIZATION { ON | OFF }`
- `ENABLE | DISABLE`
- `METHODS 'method...'`
- `SECURE { ON | OFF }`
- `USER { user-name | NULL }`

**method**:  
- `DEFAULT`
- `POST`
- `GET`
- `HEAD`
- `PUT`
- `DELETE`
- `NONE`
- `*`

**Parameters**  
- `service-name`  

Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

You can name your service **root**, but this name has a special function.
**TYPE clause**  Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

- **'RAW'**  The result set is sent to the client without any formatting. Utilization of this service requires that all content markup is explicitly provided. Complex dynamic content containing current content with markup, JavaScript and images can be generated on demand. The media type may be specified by setting the Content-Type response header using the sa_set_http_header procedure. Setting the Content-Type header to 'text/html' is good practice when generating HTML markup to ensure that all browsers display the markup as HTML and not text/plain.

- **'HTML'**  The result set is returned as an HTML representation of a table or view.

- **'JSON'**  The result set is returned in JavaScript Object Notation (JSON). A JSON service does not automatically process JSON input. It only presents data (in the response) in JSON format. JSON accepts POST/PUT methods where application/x-www-form-urlencoded is supported. If for a POST/PUT METHOD, Content-Type: application/json is specified, then the application may use http_variable('body') to retrieve the JSON (request) content. SQL Anywhere does not parse the JSON input automatically. It is up to the application to parse it. For more information about JSON, see http://www.json.org/.

- **'XML'**  The result set is returned as XML. If the result set is already XML, no additional formatting is applied. Otherwise, it is automatically formatted as XML. As an alternative approach, a RAW service could return a select using the FOR XML RAW clause having set a valid Content-Type such as text/xml using sa_set_http_header procedure.

**URL clause**  Determines whether URL paths are accepted and, if so, how they are processed. Specifying URL PATH has the same effect as URL.

- **OFF**  Indicates that the service name in a URL request must not be followed by a path. OFF is the default setting. For example, the following form is disallowed due to the path elements /aaa/bbb/ccc.

  http://host-name/service-name/aaa/bbb/ccc

  Suppose that CREATE SERVICE echo URL PATH OFF was specified when creating the web service. A URL similar to http://localhost/echo?id=1 produces the following values:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_VARIABLE('id')</td>
<td>1</td>
</tr>
<tr>
<td>HTTP_HEADER('@HttpQueryString')</td>
<td>id=1</td>
</tr>
</tbody>
</table>

- **ON**  Indicates that the service name in a URL request can be followed by a path. The path value is returned by querying a dedicated HTTP variable named URL. A service can be defined to explicitly provide the URL parameter or it may be retrieved using the HTTP_VARIABLE function. For example, the following form is allowed:

  http://host-name/service-name/aaa/bbb/ccc
Suppose that `CREATE SERVICE echo URL PATH ON` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_VARIABLE('id')</td>
<td>1</td>
</tr>
<tr>
<td>HTTP_VARIABLE('URL')</td>
<td>one/two</td>
</tr>
<tr>
<td>HTTP_HEADER('@HttpQueryString')</td>
<td>id=1</td>
</tr>
</tbody>
</table>

- **ELEMENTS** Indicates that the service name in a URL request may be followed by a path. The path is obtained in segments by specifying a single parameter keyword `URL1`, `URL2`, and so on. Each parameter may be retrieved using the `HTTP_VARIABLE` or `NEXT_HTTP_VARIABLE` functions. These iterator functions can be used in applications where a variable number of path elements can be provided. For example, the following form is allowed:

  http://host-name/service-name/aaa/bbbccc

Suppose that `CREATE SERVICE echo URL PATH ELEMENTS` was specified when creating the web service. A URL similar to `http://localhost/echo/one/two?id=1` produces the following values:

<table>
<thead>
<tr>
<th>Function call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_VARIABLE('id')</td>
<td>1</td>
</tr>
<tr>
<td>HTTP_VARIABLE('URL1')</td>
<td>one</td>
</tr>
<tr>
<td>HTTP_VARIABLE('URL2')</td>
<td>two</td>
</tr>
<tr>
<td>HTTP_VARIABLE('URL3')</td>
<td>NULL</td>
</tr>
<tr>
<td>HTTP_HEADER('@HttpQueryString')</td>
<td>id=1</td>
</tr>
</tbody>
</table>

Up to 10 elements can be obtained. A NULL value is returned if the corresponding element is not supplied. In the above example, `HTTP_VARIABLE('URL3')` returns NULL because no corresponding element was supplied.

For more information about URLs, see “How to browse a SQL Anywhere HTTP web server” [SQL Anywhere Server - Programming] and “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming].

**AUTHORIZATION clause** Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for all services and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service.
by providing a user or group name with the USER clause. If the user name is NULL, all known users can
access the service. The AUTHORIZATION clause allows your web services to use database authorization
and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic
authentication (RFC 2617) that obfuscates the user and password information using base-64 encoding. It
is recommended that you use the HTTPS protocol for increased security.

**ENABLE and DISABLE clauses** Determines whether the service is available for use. By default,
when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE
or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled
using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service
typically returns a 404 Not Found HTTP status.

**METHODS clause** Specifies the HTTP methods that are supported by the service. Valid values are
DEFAULT, POST, GET, HEAD, PUT, DELETE, and NONE. An asterisk (*) may be used as a short
form to represent the POST, GET, and HEAD methods which are default request types for the RAW,
HTML, and XML service types. Not all HTTP methods are valid for all the service types. The following
table summarizes the valid HTTP methods that can be applied to each service type:

<table>
<thead>
<tr>
<th>Method value</th>
<th>Applies to service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>all</td>
<td>Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.</td>
</tr>
<tr>
<td>POST</td>
<td>RAW, HTML, JSON, XML</td>
<td>Enabled by default.</td>
</tr>
<tr>
<td>GET</td>
<td>RAW, HTML, JSON, XML</td>
<td>Enabled by default.</td>
</tr>
<tr>
<td>HEAD</td>
<td>RAW, HTML, JSON, XML</td>
<td>Enabled by default.</td>
</tr>
<tr>
<td>PUT</td>
<td>RAW, HTML, JSON, XML</td>
<td>Not enabled by default.</td>
</tr>
<tr>
<td>DELETE</td>
<td>RAW, HTML, JSON, XML</td>
<td>Not enabled by default.</td>
</tr>
<tr>
<td>NONE</td>
<td>all</td>
<td>Use NONE to disable access to a service.</td>
</tr>
<tr>
<td>*</td>
<td>RAW, HTML, JSON, XML</td>
<td>Same as specifying 'POST,GET,HEAD'.</td>
</tr>
</tbody>
</table>

For example, you can use either of the following clauses to specify that a service supports all HTTP
method types:
To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

**SECURE clause** Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

**USER clause** Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a 401 Authorization Required HTTP response status. Based on this response, the web browser prompts for a user ID and password.

**Caution**
It is strongly recommended that you specify a USER clause when authorization is enabled (default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

**Statement** Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

An HTTP request to a non-DISH service with no statement specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use statement to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following statement demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

For more information about passing host variables to a function or procedure, see “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming].

**Remarks**
Service definitions are stored within the ISYSWEBSERVICE table and can be examined from the SYSWEBSERVICE system view.

If a USER clause is specified, the service is dropped if user-name is dropped.
Privileges
You must have the MANAGE ANY WEB SERVICE system privilege.

Side effects
None.

See also
- “ALTER SERVICE statement [HTTP web service]” on page 466
- “DROP SERVICE statement” on page 772
- “sp_parse_json system procedure” on page 1295
- “SYSWEBSERVICE system view” on page 1418
- “SQL Anywhere as an HTTP web server” [SQL Anywhere Server - Programming]
- “How to create and customize a root web service” [SQL Anywhere Server - Programming]
- “sa_set_http_header system procedure” on page 1239
- “How to develop web service applications in an HTTP web server” [SQL Anywhere Server - Programming]
- “Identifiers” on page 4
- “ROW constructor [Composite]” on page 351
- “ARRAY constructor [Composite]” on page 157

Standards and compatibility
- SQL/2008 Vendor extension.
- Transact-SQL CREATE SERVICE is supported by Adaptive Server Enterprise, for XML and RAW types only.

Examples
The following example demonstrates how to create a JSON service.

Start a database server with the -xs (http or https) option and then execute the following SQL statements to set up the service:

```
CREATE PROCEDURE ListEmployees()
RESULT (  
EmployeeID integer,  
Surname person_name_t,  
GivenName person_name_t,  
StartDate date,  
TerminationDate date )
BEGIN
    SELECT EmployeeID, Surname, GivenName, StartDate, TerminationDate
    FROM GROUPO.Employees
END;

CREATE SERVICE "jsonEmployeeList"
TYPE 'JSON' 
AUTHORIZATION OFF 
SECURE OFF 
USER DBA 
AS CALL ListEmployees();
```

The JSON service provides data for easy consumption by an AJAX call back.
Execute the following SQL statement to create an HTML service that provides the service in a readable form:

```sql
CREATE SERVICE "EmployeeList"
  TYPE 'HTML'
  AUTHORIZATION OFF
  SECURE OFF
  USER DBA
  AS CALL ListEmployees();
```

Use a web browser to access the service using a URL similar to http://localhost/EmployeeList.

**CREATE SERVICE statement [SOAP web service]**

Creates a new SOAP over HTTP or DISH service.

**Syntax 1 - SOAP over HTTP services**

```sql
CREATE SERVICE service-name
  TYPE 'SOAP'
  DATATYPE { ON | OFF | IN | OUT }
  FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL }
  common-attributes
  AS statement

common-attributes:
  AUTHORIZATION { ON | OFF }
  ENABLE | DISABLE
  METHODS 'method,...'
  SECURE { ON | OFF }
  USER { user-name | NULL }

method:
  DEFAULT
  POST
  HEAD
  NONE
```

**Syntax 2 - DISH services**

```sql
CREATE SERVICE service-name
  TYPE 'DISH'
  GROUP { group-name | NULL }
  FORMAT { 'DNET' | 'CONCRETE' [ EXPLICIT { ON | OFF } ] | 'XML' | NULL }
  common-attributes

common-attributes:
  AUTHORIZATION { ON | OFF }
  ENABLE | DISABLE
  METHODS 'method,...'
  SECURE { ON | OFF }
  USER { user-name | NULL }

method:
  DEFAULT
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
POST
GET
HEAD
NONE

Parameters

**service-name**  Web service names can be any sequence of alphanumeric characters or slash (/), hyphen (-), underscore (_), period (.), exclamation mark (!), tilde (~), asterisk (*), apostrophe ('), left parenthesis ((), or right parenthesis ()), except that the service name must not begin or end with a slash (/) or contain two or more consecutive slashes (for example, //).

Unlike other services, you cannot use a slash (/) anywhere in a DISH service name.

You can name your service **root**, but this name has a special function.

**TYPE clause**  Identifies the type of the service where each service defines a specific response format. The type must be one of the listed service types. There is no default value.

- **'SOAP'**  The result set is returned as an XML payload known as a SOAP envelope. The format of the data may be further refined using by the FORMAT clause. A request to a SOAP service must be a valid SOAP request, not just a general HTTP request. For more information about the SOAP standards, see [http://www.w3.org/TR/2000/NOTE-SOAP-20000508/](http://www.w3.org/TR/2000/NOTE-SOAP-20000508/).

- **'DISH'**  A DISH service (Determine SOAP Handler) is a SOAP endpoint that references any SOAP service within its GROUP context. It also exposes the interfaces to its SOAP services by generating a WSDL (Web Services Description Language) for consumption by SOAP client toolkits.

**GROUP clause**  A DISH service without a GROUP clause exposes all SOAP services defined within the database. By convention, the SOAP service name can be composed of a GROUP and a NAME element. The name is delimited from the group by the last slash character. For example, a SOAP service name defined as **'aaa/bbb/cdd'** is **'cdd'**, and the group is **'aaa/bbb'**. Delimiting a DISH service using this convention is invalid. Instead, a GROUP clause is applied to specify the group of SOAP services for which it is to be the SOAP endpoint.

**Note**

Slashes are converted to underscores within the WSDL to produce valid XML. Use caution when using a DISH service that does not specify a GROUP clause such that it exposes all SOAP services that may contain slashes. Use caution when using groups with SOAP service names that contain underscores to avoid ambiguity.

**DATATYPE clause**  Applies to SOAP services only. When DATATYPE OFF is specified, SOAP input parameters and response data are defined as XMLSchema string types. In most cases, true data types are preferred because it negates the need for the SOAP client to cast the data prior to computation. Parameter data types are exposed in the schema section of the WSDL generated by the DISH service. Output data types are represented as XML schema type attributes for each column of data.

The following values are permitted for the DATATYPE clause:
Generates data typing of input parameters and result set responses.

- **ON**
- **OFF** All input parameters and response data are typed as **XMLSchema** string (default).
- **IN** Generates true data types for input parameters only. Response data types are **XMLSchema** string.
- **OUT** Generates true data types for responses only. Input parameters are typed as **XMLSchema** string.

For more information about SOAP services, see “Tutorial: Using SQL Anywhere to access a SOAP/DISH service” [SQL Anywhere Server - Programming].

For more information about mapping XMLSchema types to SQL data types, see “SOAP data types” [SQL Anywhere Server - Programming].

**FORMAT clause** This clause specifies the output format when sending responses to SOAP client applications.

The SOAP service format is dictated by the associated DISH service format specification when it is not specified by the SOAP service. The default format is DNET.

SOAP requests should be directed to the DISH service (the SQL Anywhere SOAP endpoint) to leverage common formatting rules for a group of SOAP services (SOAP operations). A SOAP service FORMAT specification overrides that of a DISH service. The format specification of the DISH service is used when a SOAP service does not define a FORMAT clause. If no FORMAT is provided by either service then the default is **'DNET'**.

The following formats are supported:

- **'DNET'** The output is in a System.Data.DataSet compatible format for consumption by .NET client applications. (default)

- **'CONCRETE'** This output format is used to support client SOAP toolkits that are capable of generating interfaces representing arrays of row and column objects but are not able to consume the DNET format. Java and .NET clients can easily consume this output format.

  The specific format is exposed within the WSDL as an explicit dataset object or a **SimpleDataset**. Both dataset representations describe a data structure representing an array of rows with each row containing an array of columns. An explicit dataset object has the advantage of representing the actual shape of the result set by providing parameter names and datatypes for each column in the row. In contrast, the **SimpleDataset** exposes rows containing an unbounded number of columns of any type.

  **FORMAT 'CONCRETE' EXPLICIT ON** requires that the Service statement calls a stored procedure which defines a RESULT clause. Having met this condition, the SOAP service will expose an explicit dataset whose name begins with the service name appended with **Dataset**.

  If the condition is not met, a **SimpleDataset** is used.

- **'XML'** The output is generated in an XMLSchema string format. The response is an XML document that requires further processing by the SOAP client to extract column data. This format is suitable for
SOAP clients that cannot generate intermediate interface objects that represent arrays of rows and columns.

- **NULL**  A NULL type causes the SOAP or DISH service to use the default behavior. The format type of an existing service is overwritten when using the NULL type in an ALTER SERVICE statement.

**AUTHORIZATION clause**  Determines whether users must specify a user name and password through basic HTTP authorization when connecting to the service. The default value is ON. If authorization is OFF, the AS clause is required for SOAP services, and a user must be specified with the USER clause. All requests are run using that user's account and privileges. If AUTHORIZATION is ON, all users must provide a user name and password. Optionally, you can limit the users that are permitted to use the service by providing a user or group name with the USER clause. If the user name is NULL, all known users can access the service. The AUTHORIZATION clause allows your web services to use database authorization and privileges to control access to the data in your database.

When the authorization value is ON, an HTTP client connecting to a web service uses basic authentication (RFC 2617) which obfuscates the user and password information using base-64 encoding. It is recommended that you use the HTTPS protocol for increased security.

**ENABLE and DISABLE clauses**  Determines whether the service is available for use. By default, when a service is created, it is enabled. When creating or altering a service, you may include an ENABLE or DISABLE clause. Disabling a service effectively takes the service off line. Later, it can be enabled using ALTER SERVICE with the ENABLE clause. An HTTP request made to a disabled service typically returns a 404 Not Found HTTP status.

**METHODS clause**  Specifies the HTTP methods that are supported by the service. Valid values are DEFAULT, POST, GET, HEAD, and NONE. An asterisk (*) may be used as a short form to represent the POST, GET, and HEAD methods. The default method types for SOAP services are POST and HEAD. The default method types for DISH services are GET, POST, and HEAD. Not all HTTP methods are valid for all the service types. The following table summarizes the valid HTTP methods that can be applied to each service type:

<table>
<thead>
<tr>
<th>Method value</th>
<th>Applies to service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>both</td>
<td>Use DEFAULT to reset the set of default HTTP methods for the given service type. It cannot be included in a list with other method values.</td>
</tr>
<tr>
<td>POST</td>
<td>both</td>
<td>Enabled by default for SOAP.</td>
</tr>
<tr>
<td>GET</td>
<td>DISH only</td>
<td>Enabled by default for DISH.</td>
</tr>
<tr>
<td>HEAD</td>
<td>both</td>
<td>Enabled by default for SOAP and DISH.</td>
</tr>
<tr>
<td>Method value</td>
<td>Applies to service</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NONE</td>
<td>both</td>
<td>Use NONE to disable access to a service. When applied to a SOAP service, the service cannot be directly accessed by a SOAP request. This enforces exclusive access to a SOAP operation through a DISH service SOAP endpoint. It is recommended that you specify METHOD NONE for each SOAP service.</td>
</tr>
<tr>
<td>*</td>
<td>DISH only</td>
<td>Same as specifying 'POST,GET,HEAD'.</td>
</tr>
</tbody>
</table>

For example, you can use the following clause to specify that a service supports all SOAP over HTTP method types:

```
METHODS 'POST,HEAD'
```

To reset the list of request types for any service type to its default, you can use the following clause:

```
METHODS 'DEFAULT'
```

**SECURE clause**  Specifies whether the service should be accessible on a secure or non-secure listener. ON indicates that only HTTPS connections are accepted, and that connections received on the HTTP port are automatically redirected to the HTTPS port. OFF indicates that both HTTP and HTTPS connections are accepted, provided that the necessary ports are specified when starting the web server. The default value is OFF.

**USER clause**  Specifies a database user, or group of users, with privileges to execute the web service request. A USER clause must be specified when the service is configured with AUTHORIZATION OFF and should be specified with AUTHORIZATION ON (the default). An HTTP request made to a service requiring authorization results in a 401 Authorization Required HTTP response status. Based on this response, the web browser prompts for a user ID and password.

**Caution**  It is strongly recommended that you specify a USER clause when authorization is enabled (the default). Otherwise, authorization is granted to all users.

The USER clause controls which database user accounts can be used to process service requests. Database access permissions are restricted to the privileges assigned to the user of the service.

**statement**  Specifies a command, such as a stored procedure call, to invoke when the service is accessed.

A DISH service is the only service that must either define a null statement, or not define a statement. A SOAP service must define a statement. Any other SERVICE can have a NULL statement, but only if it is configured with AUTHORIZATION ON.
An HTTP request to a non-DISH service with no statement specifies the SQL expression to execute within its URL. Although authorization is required, this capability should not be used in production systems because it makes the server vulnerable to SQL injections. When a statement is defined within the service, the specified SQL statement is the only statement that can be executed through the service.

In a typical web service application, you use statement to call a function or procedure. You can pass host variables as parameters to access client-supplied HTTP variables.

The following statement demonstrates a procedure call that passes two host variables to a procedure named AuthenticateUser. This call presumes that a web client supplies the user_name and user_password variables:

```
CALL AuthenticateUser ( :user_name, :user_password );
```

For more information about passing host variables to a function or procedure, see “How to access client-supplied HTTP variables and headers” [SQL Anywhere Server - Programming].

Remarks

Service definitions are stored within the ISYSWEBSERVICE table and can be examined from the SYSWEBSERVICE view.

Privileges

You must have the MANAGE ANY WEB SERVICE system privilege.

Side effects

None.

See also

- “ALTER SERVICE statement [SOAP web service]” on page 471
- “DROP SERVICE statement” on page 772
- “SYSWEBSERVICE system view” on page 1418
- “SQL Anywhere as an HTTP web server” [SQL Anywhere Server - Programming]
- “How to create and customize a root web service” [SQL Anywhere Server - Programming]

Standards and compatibility

- **SQL/2008** Vendor extension.
- **Transact-SQL** CREATE SERVICE is supported by Adaptive Server Enterprise for SOAP types only.

**CREATE SPATIAL REFERENCE SYSTEM statement**

Creates or replaces a spatial reference system.

Syntax

```
{ CREATE [ OR REPLACE ] SPATIAL REFERENCE SYSTEM
| CREATE SPATIAL REFERENCE SYSTEM IF NOT EXISTS }
```
srs-name
[ srs-attribute ] [ srs-attribute ... ]

srs-name : string

srs-attribute :
Identified by srs-id
| Definition { definition-string | NULL } |
| Organization { organization-name Identifier by organization-srs-id | NULL } |
| Transform definition { transform-definition-string | NULL } |
| Linear unit of measure { linear-unit-name } |
| Angular unit of measure { angular-unit-name | NULL } |
| Type { Round Earth | Planar } |
| Coordinate coordinate-name { Unbounded | Between low-number and high-number } |
| Ellipsoid semi major axis semi-major-axis-length | Semi minor axis semi-minor-axis-length |
| Inverse flattening inverse-flattening-ratio |
| Snap to grid { grid-size | DEFAULT } |
| Tolerance { tolerance-distance | DEFAULT } |
| Polygon format polygon-format |
| Storage format storage-format |

srs-id : integer

semi-major-axis-length : number

semi-minor-axis-length : number

inverse-flattening-ratio : number

gird-size : DOUBLE : usually between 0 and 1

tolerance-distance : number

axis-order : { 'x/y/z/m' | 'long/lat/z/m' | 'lat/long/z/m' }

polygon-format : { 'CounterClockWise' | 'Clockwise' | 'EvenOdd' }

storage-format : { 'Internal' | 'Original' | 'Mixed' }

Parameters

OR REPLACE clause Specifying OR REPLACE creates the spatial reference system if it does not already exist in the database, and replaces it if it does exist. An error is returned if you attempt to replace a spatial reference system while it is in use. An error is also returned if you attempt to replace a spatial reference system that already exists in the database without specifying the OR REPLACE clause.

CREATE SPATIAL REFERENCE IF NOT EXISTS Specifying CREATE SPATIAL REFERENCE IF NOT EXISTS checks to see if a spatial reference system by that name already exists. If it does not exist, the database server creates the spatial reference system. If it does exist, no further action is performed and no error is returned.

IDENTIFIED BY clause Use this clause to specify the SRID (srs-id) for the spatial reference system. If the spatial reference system is defined by an organization with an organization-srs-id, then srs-id should be set to that value.
If the IDENTIFIED BY clause is not specified, then the SRID defaults to the organization-srs-id defined by either the ORGANIZATION clause or the DEFINITION clause. If neither clause defines an organization-srs-id that could be used as a default SRID, an error is returned.

When the spatial reference system is based on a well known coordinate system, but has a different geodesic interpretation, set the srs-id value to be 1000000000 (one billion) plus the well known value. For example, the SRID for a planar interpretation of the geodetic spatial reference system WGS 84 (ID 4326) would be 1000004326.

With the exception of SRID 0, spatial reference systems provided by SQL Anywhere that are not based on well known systems are given a SRID of 2000000000 (two billion) and above. The range of SRID values from 2000000000 to 2147483647 is reserved by SQL Anywhere and you should not create SRIDs in this range.

To reduce the possibility of choosing a SRID that is reserved by a defining authority such as OGC or by other vendors, you should not choose a SRID in the range 0 - 32767 (reserved by EPSG), or in the range 2147483547 - 2147483647.

Also, since the SRID is stored as a signed 32-bit integer, the number cannot exceed $2^{31}-1$ or 2147483647.

**DEFINITION clause**  Use this clause to set, or override, default coordinate system settings. If any attribute is set in a clause other than the DEFINITION clause, it takes the value specified in the other clause regardless of what is specified in the DEFINITION clause.

definition-string is a string in the Spatial Reference System Well Known Text syntax as defined by SQL/MM and OGC. For example, the following query returns the definition for WGS 84.

```sql
SELECT ST_SpatialRefSys::ST_FormatWKT( definition )
FROM ST_SPATIAL_REFERENCE_SYSTEMS
WHERE srs_id=4326;
```

In Interactive SQL, if you double-click the value returned, an easier to read version of the value appears.

When the DEFINITION clause is specified, definition-string is parsed and used to choose default values for attributes. For example, definition-string may contain an AUTHORITY element that defines the organization-name and organization-srs-id.

Parameter values in definition-string are overridden by values explicitly set using the SQL statement clauses. For example, if the ORGANIZATION clause is specified, it overrides the value for ORGANIZATION in definition-string.

**ORGANIZATION clause**  Use this clause to specify information about the organization that created the spatial reference system that the new spatial reference system is based on. organization-name is the name of the organization that created it; organization-srs-id is the numeric identifier the organization uses to identify the spatial reference system.

**TRANSFORM DEFINITION clause**  Use this clause to specify a description of the transform to use for the spatial reference system. Currently, only the PROJ.4 transform is supported. For example, the transform-definition-string for WGS 84 is '+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs'.

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 675
If you specify an unsupported transform definition, an error is returned.

The transform definition is used by the ST_Transform method when transforming data between spatial reference systems. Some transforms may still be possible even if there is no transform-definition-string defined.

**LINEAR UNIT OF MEASURE clause**  Use this clause to specify the linear unit of measure for the spatial reference system. The value you specify must match a linear unit of measure that is defined in the ST_UNITS_OF_MEASURE system view.

If this clause is not specified, and is not defined in the DEFINITION clause, the default is METRE.

To add predefined units of measure to the database, use the sa_install_feature system procedure.

To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

---

**Note**

While both METRE and METER are accepted spellings, METRE is preferred as it conforms to the SQL/MM standard.

---

**ANGULAR UNIT OF MEASURE clause**  Use this clause to specify the angular unit of measure for the spatial reference system. The value you specify must match an angular unit of measure defined in the ST_UNITS_OF_MEASURE system table.

If this clause is not specified, and is not defined in the DEFINITION clause, the default is DEGREE for geographic spatial reference systems and NULL for non-geographic spatial reference systems.

The angular unit of measure must be non-NULL for geographic spatial reference systems and it must be NULL for non-geographic spatial reference systems.

To add predefined units of measure to the database, use the sa_install_feature system procedure.

To add custom units of measure to the database, use the CREATE SPATIAL UNIT OF MEASURE statement.

**TYPE clause**  Use the TYPE clause to control how the SRS interprets lines between points. For geographic spatial reference systems, the TYPE clause can specify either ROUND EARTH (the default) or PLANAR. The ROUND EARTH model interprets lines between points as great elliptic arcs. Given two points on the surface of the Earth, a plane is selected that intersects the two points and the center of the Earth. This plane intersects the Earth, and the line between the two points is the shortest distance along this intersection.

For two points that lie directly opposite each other, there is not a single unique plane that intersects the two points and the center of the Earth. Line segments connecting these antipodal points are not valid and give an error in the ROUND EARTH model.

The ROUND EARTH model treats the Earth as a spheroid and selects lines that follow the curvature of the Earth. In some cases, it may be necessary to use a planar model where a line between two points is interpreted as a straight line in the equirectangular projection where x=long, y=lat.
In the following example, the blue line shows the line interpretation used in the ROUND EARTH model and the red line shows the corresponding PLANAR model.

The PLANAR model may be used to match the interpretation used by other products. The PLANAR model may also be useful because there are some limitations for methods that are not supported in the ROUND EARTH model (such as ST_Area, ST_ConvexHull) and some are partially supported (ST_Distance only supported between point geometries). Geometries based on circularstrings are not supported in ROUND EARTH spatial reference systems.

For non-geographic SRSs, the type must be PLANAR (and that is the default if the TYPE clause is not specified and either the DEFINITION clause is not specified or it uses a non-geographic definition).

**COORDINATE clause**  Use this clause to specify the bounds on the spatial reference system's dimensions. *coordinate-name* is the name of the coordinate system used by the spatial reference system. For non-geographic coordinate systems, *coordinate-name* can be x, y, or m. For geographic coordinate systems, *coordinate-name* can be LATITUDE, LONGITUDE, z, or m.

Specify UNBOUNDED to place no bounds on the dimensions. Use the BETWEEN clause to set low and high bounds.

The X and Y coordinates must have associated bounds. For geographic spatial reference systems, the longitude coordinate is bounded between -180 and 180 degrees and the latitude coordinate is bounded between -90 and 90 degrees by default the unless COORDINATE clause overrides these settings. For non-geographic spatial reference systems, the CREATE statement must specify bounds for both X and Y coordinates.

LATITUDE and LONGITUDE are used for geographic coordinate systems. The bounds for LATITUDE and LONGITUDE default to the entire Earth, if not specified.
**ELLIPSOID clause**  Use the ellipsoid clause to specify the values to use for representing the Earth as an ellipsoid for spatial reference systems of type ROUND EARTH. If the DEFINITION clause is present, it can specify ellipsoid definition. If the ELLIPSOID clause is specified, it overrides this default ellipsoid.

The Earth is not a perfect sphere because the rotation of the Earth causes a flattening so that the distance from the center of the Earth to the North or South pole is less than the distance from the center to the equator. For this reason, the Earth is modeled as an ellipsoid with different values for the semi-major axis (distance from center to equator) and semi-minor axis (distance from center to the pole). It is most common to define an ellipsoid using the semi-major axis and the inverse flattening, but it can instead be specified using the semi-minor axis (for example, this approach must be used when a perfect sphere is used to approximate the Earth). The semi-major and semi-minor axes are defined in the linear units of the spatial reference system, and the inverse flattening \((1/f)\) is a ratio:

\[
1/f = \frac{\text{semi-major-axis}}{\text{semi-major-axis} - \text{semi-minor-axis}}
\]

SQL Anywhere uses the ellipsoid definition when computing distance in geographic spatial reference systems.

The ellipsoid must be defined for geographic spatial reference systems (either in the DEFINITION clause or the ELLIPSOID clause), and it must not be specified for non-geographic spatial reference systems.

**SNAP TO GRID clause**  For flat-Earth (planar) spatial reference systems, use the SNAP TO GRID clause to define the size of the grid SQL Anywhere uses when performing calculations. By default, SQL Anywhere selects a grid size so that 12 significant digits can be stored at all points in the space bounds for X and Y. For example, if a spatial reference system bounds X between -180 and 180 and Y between -90 and 90, then a grid size of 0.000000001 (1E-9) is selected.

\(\text{grid-size}\) must be large enough so that points snapped to the grid can be represented with equal precision at all points in the bounded space. If \(\text{grid-size}\) is too small, the server reports an error.

When set to 0, no snapping to grid is performed.

For round-Earth spatial reference systems, SNAP TO GRID must be set to 0.

Specify SNAP TO GRID DEFAULT to set the grid size to the default that the database server would use.

**TOLERANCE clause**  For flat-Earth (planar) spatial reference systems, use the TOLERANCE clause to specify the precision to use when comparing points. If the distance between two points is less than \(\text{tolerance-distance}\), the two points are considered equal. Setting \(\text{tolerance-distance}\) allows you to control the tolerance for imprecision in the input data or limited internal precision. By default, \(\text{tolerance-distance}\) is set to be equal to \(\text{grid-size}\).

When set to 0, two points must be exactly equal to be considered equal.

For round-Earth spatial reference systems, TOLERANCE must be set to 0.

**POLYGON FORMAT clause**  Internally, SQL Anywhere interprets polygons by looking at the orientation of the constituent rings. As one travels a ring in the order of the defined points, the inside of the polygon is on the left side of the ring. The same rules are applied in PLANAR and ROUND EARTH spatial reference systems.
The interpretation used by SQL Anywhere is a common but not universal interpretation. Some products use the exact opposite orientation, and some products do not rely on ring orientation to interpret polygons. The POLYGON FORMAT clause can be used to select a polygon interpretation that matches the input data, as needed. The following values are supported:

- **'CounterClockwise'** The input follows SQL Anywhere's internal interpretation: the inside of the polygon is on the left side while following ring orientation.

- **'Clockwise'** The input follows the opposite of SQL Anywhere's approach: the inside of the polygon is on the right side while following ring orientation.

- **'EvenOdd'** EvenOdd is the default format. With EvenOdd, the orientation of rings is ignored and the inside of the polygon is instead determined by looking at the nesting of the rings, with the exterior ring being the largest ring and interior rings being smaller rings inside this ring. A ray is traced from a point within the rings and radiating outward crossing all rings. If the number the ring being crossed is an even number, it is an outer ring. If it is odd, it is an inner ring.

### STORAGE FORMAT clause

When you insert spatial data into the database from an external format (such as WKT or WKB), the database server normalizes the data to improve the performance and semantics of spatial operations. The normalized representation may differ from the original representation (for example, in the orientation of polygon rings or the precision stored in individual coordinates). While spatial equality is maintained after the normalization, some original input characteristics may not be reproducible, such as precision and ring orientation. In some cases you may want to store the original representation, either exclusively, or in addition to the normalized representation.

To control what is stored, specify the STORAGE FORMAT clause followed by one of the following values:

- **'Internal'** SQL Anywhere stores only the normalized representation. Specify this value when the original input characteristics do not need to be reproduced. This is the default for planar spatial reference systems (TYPE PLANAR).

  **Note**

  If you are using MobiLink to synchronize your spatial data, you should specify **Mixed**. MobiLink tests for equality during synchronization, which requires the data in its original format.

- **'Original'** SQL Anywhere stores only the original representation. The original input characteristics can be reproduced, but all operations on the stored values must repeat normalization steps, possibly slowing down operations on the data.

- **'Mixed'** SQL Anywhere stores the internal version and, if it is different from the original version, it stores the original version as well. By storing both versions, the original representation characteristics can be reproduced and operations on stored values do not need to repeat normalization steps. However, storage requirements may increase significantly because potentially two representations are being stored for each geometry.

  Mixed is the default format for round-Earth spatial reference systems (TYPE ROUND EARTH).
Remarks

For a geographic spatial reference system, you can specify both a LINEAR and an ANGULAR unit of measure; otherwise for non-geographic, you specify only a LINEAR unit of measure. The LINEAR unit of measure is used for computing distance between points and areas. The ANGULAR unit of measure tells how the angular latitude/longitude are interpreted and is NULL for projected coordinate systems, non-NULL for geographic coordinate systems.

All derived geometries returned by operations are normalized.

When working with data that is being synchronized with a non-SQL Anywhere database, STORAGE FORMAT should be set to either 'Original' or 'Mixed' so that the original characteristics of the data can be preserved.

Privileges

You must have the MANAGE ANY SPATIAL OBJECT or CREATE ANY OBJECT system privilege.

Side effects

None

See also

- “sa_install_feature system procedure” on page 1164
- “CREATE SPATIAL UNIT OF MEASURE statement” on page 681
- “ST_UNITS_OF_MEASURE consolidated view” on page 1423
- “ST_SPATIAL_REFERENCESYSTEMS consolidated view” on page 1420
- “ALTER SPATIAL REFERENCE SYSTEM statement” on page 477
- “Spatial reference systems (SRS) and Spatial reference identifiers (SRID)” [SQL Anywhere Server - Spatial Data Support]
- “Spatial data” [SQL Anywhere Server - Spatial Data Support]
- “sa_install_feature system procedure” on page 1164

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example creates a spatial reference system named mySpatialRS:

```
CREATE SPATIAL REFERENCE SYSTEM  "mySpatialRS"
IDENTIFIED BY 1000026980
LINEAR UNIT OF MEASURE "metre"
TYPE PLANAR
COORDINATE X BETWEEN 171266.736269555 AND 831044.757769222
COORDINATE Y BETWEEN 524881.608973277 AND 691571.125115319
DEFINITION 'PROJCS["NAD83 / Kentucky South",
GEOGCS["NAD83",
DATUM["North_American_Datum_1983",
SPHEROID["GRS 1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],
AUTHORITY["EPSG","6269"]],
PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
UNIT["degree",0.01745329251994328,AUTHORITY["EPSG","9122"]],
AUTHORITY["EPSG","4269"]],
UNIT["metre",1,AUTHORITY["EPSG","9001"]],
```
CREATE SPATIAL UNIT OF MEASURE statement

Creates or replaces a spatial unit of measurement.

Syntax

```
CREATE [ OR REPLACE ] SPATIAL UNIT OF MEASURE identifier
   TYPE { LINEAR | ANGULAR }
   [ CONVERT USING number ]
```

Parameters

**OR REPLACE clause**   Including the OR REPLACE creates a new spatial unit of measure, or replaces an existing spatial unit of measure with the same name. This clause preserves existing privileges. An error is returned if you attempt to replace a spatial unit that is already in use.

**TYPE clause**   Defines whether the unit of measure is used for angles (ANGULAR) or distances (LINEAR).

**CONVERT USING**   The conversion factor for the spatial unit relative to the base unit. For linear units, the base unit is METRE. For angular units, the base unit is RADIANT.

Remarks

The CONVERT USING clause is used to define how to convert a measurement in the defined unit of measure to the base unit of measure (radians or meters). The measurement is multiplied by the supplied conversion factor to get a value in the base unit of measure. For example, a measurement of 512 millimeters would be multiplied by a conversion factor of 0.001 to get a measurement of 0.512 metres.

Spatial reference systems always include a linear unit of measure to be used when calculating distances (ST_Distance or ST_Length), or area. For example, if the linear unit of measure for a spatial reference system is miles, then the area unit used is square miles. In some cases, spatial methods accept an optional parameter that specifies the linear unit of measure to use. For example, if the linear unit of measure for a spatial reference system is in miles, you could retrieve the distance between two geometries in meters by using the optional parameter 'metre'.

For projected coordinate systems, the X and Y coordinates are specified in the linear unit of the spatial reference system. For geographic coordinate systems, the latitude and longitude are specified in the angular units of measure associated with the spatial reference system. In many cases, this angular unit of measure is degrees but any valid angular unit of measure can be used.
You can use the sa_install_feature system procedure to add predefined units of measure to your database.

**Privileges**

You must have the MANAGE ANY SPATIAL OBJECT or CREATE ANY OBJECT system privilege.

**Side effects**

None

**See also**

- “sa_install_feature system procedure” on page 1164
- “DROP SPATIAL UNIT OF MEASURE statement” on page 774
- “Spatial data” [SQL Anywhere Server - Spatial Data Support]

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following example creates a spatial unit of measure named Test.

```
CREATE SPATIAL UNIT OF MEASURE Test
   TYPE LINEAR
   CONVERT USING 15;
```

**CREATE STATISTICS statement**

Recreates the column statistics used by the optimizer, and stores them in the ISYSCOLSTAT system table.

**Syntax**

```
CREATE STATISTICS object-name [ ( column-list ) ]
```

```
object-name:
   table-name | materialized-view-name | temp-table-name
```

**Remarks**

The CREATE STATISTICS statement recreates the column statistics that SQL Anywhere uses to optimize database queries, and can be performed on base tables, materialized views, local temporary tables, and global temporary tables. You cannot create statistics on proxy tables. Column statistics include histograms, which reflect the distribution of data in the database for the specified columns. By default, column statistics are automatically created for tables with five or more rows.

In rare circumstances, when your database queries are very variable, and when data distribution is not uniform or the data is changing frequently, you can improve performance by executing the CREATE STATISTICS statement on a table or column.

When executing, the CREATE STATISTICS statement updates existing column statistics regardless of the size of the table, unless the table is empty, in which case nothing is done. If column statistics exist for
an empty table, they remain unchanged by the CREATE STATISTICS statement. To remove column statistics for an empty table, execute the DROP STATISTICS statement.

The process of executing CREATE STATISTICS performs a complete scan of the table. For this reason, careful consideration should be made before executing a CREATE STATISTICS statement.

If you drop statistics, it is recommended that you recreate them using the CREATE STATISTICS statement. Without statistics, the optimizer can generate inefficient data access plans, causing poor database performance.

**Privileges**

You must be the table owner, or have the MANAGE ANY STATISTICS or CREATE ANY OBJECT system privilege.

**Side effects**

Execution plans may change.

**See also**

- “Optimizer estimates and statistics” [SQL Anywhere Server - SQL Usage]
- “DROP STATISTICS statement” on page 775
- “LOAD TABLE statement” on page 873
- “SYSCOLSTAT system view” on page 1351
- “Histogram utility (dbhist)” [SQL Anywhere Server - Database Administration]
- “sa_get_histogram system procedure” on page 1145

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statement updates the column statistics for the ProductID column of the SalesOrderItems table:

```
CREATE STATISTICS GROUPO.SalesOrderItems ( ProductID );
```

**CREATE SUBSCRIPTION statement [SQL Remote]**

Creates a subscription for a user to a publication.

**Syntax**

```
CREATE SUBSCRIPTION
TO publication-name [ ( subscription-value ) ]
FOR subscriber-id

publication-name : identifier

subscription-value : string

subscriber-id : string
```
Parameters

**publication-name**  The name of the publication to which the user is being subscribed. This can include the owner of the publication.

**subscription-value**  A string that is compared to the subscription expression of the publication. The subscriber receives all rows for which the subscription expression matches the subscription value.

**subscriber-id**  The user ID of the subscriber to the publication. At the consolidated database, when you create a subscription to a remote user, the remote user must have been granted REMOTE privilege. At the remote database, when you create a subscription to the consolidated user, that user must have been granted CONSOLIDATED privilege.

Remarks

In SQL Remote, publications and subscriptions are two-way relationships. If you create a subscription for a remote user to a publication on a consolidated database, you should also create a subscription for the consolidated user to a publication on the remote database. By default, the Extraction utility (dbxtract) and the Extract Database Wizard grant the appropriate PUBLISH and CONSOLIDATE privilege to users in the remote databases.

If **subscription-value** is supplied, it is matched against each SUBSCRIBE BY expression in the publication. The subscriber receives all rows for which the value of the expression is equal to the supplied string.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

See also

- “DROP SUBSCRIPTION statement [SQL Remote]” on page 776
- “GRANT REMOTE statement [SQL Remote]” on page 844
- “GRANT CONSOLIDATE statement [SQL Remote]” on page 834
- “SYNCHRONIZE SUBSCRIPTION statement [SQL Remote]” on page 1016
- “START SUBSCRIPTION statement [SQL Remote]” on page 1000
- “GRANT PUBLISH statement [SQL Remote]” on page 839
- “SYSSUBSCRIPTION system view” on page 1395
- “Subscriptions” [SQL Remote]
- “Replicating the primary key pool” [SQL Remote]
- “Publish only some rows in a table” [SQL Remote]
- “Publishing only some rows using the SUBSCRIBE BY clause” [SQL Remote]
- “Publishing only some rows using a WHERE clause” [SQL Remote]

Standards and compatibility

- SQL/2008  Vendor extension.
Example

The following statement creates a subscription for the user p_chin to the publication pub_sales. The subscriber receives all rows for which the subscription expression has a value of Eastern.

```
CREATE SUBSCRIPTION
TO pub_sales ( 'Eastern' )
FOR p_chin;
```

The following statement creates a subscription for user name SamS to the CustomerPub publication, which was created using a WHERE clause:

```
CREATE SUBSCRIPTION
TO CustomerPub
FOR Sam_Singer;
```

The following statement creates a subscription for user name SamS to the PubOrders publication, defined with a subscription expression SalesRepresentative, requesting the rows for Sam Singer's own sales:

```
CREATE SUBSCRIPTION
TO PubOrders ( '856' )
FOR Sam_Singer;
```

CREATE SYNCHRONIZATION PROFILE statement

[MobiLink]

Creates a SQL Anywhere synchronization profile.

Syntax

```
CREATE [ OR REPLACE ] SYNCHRONIZATION PROFILE name string
```

Parameters

- **OR REPLACE clause** Specify CREATE OR REPLACE SYNCHRONIZATION PROFILE replaces the definition of the named synchronization profile if it already exists.
- **name** Specifies the name of the synchronization profile to create. Each profile must have a unique name.
- **string** Specify a valid option string as described below. Option strings are specified as semicolon delimited lists of elements of the form **option-name=option-value**. For example subscription=s1;verbosity=high.

Remarks

Synchronization profiles are named collections of synchronization options that can be used to control synchronization. For a list of the synchronization profile options supported by dbmlsync, see “MobiLink synchronization profiles” [MobiLink - Client Administration].

For options that take a Boolean value, setting the value to TRUE is equivalent to specifying the corresponding option on the command line.

The following values can be used to specify TRUE: TRUE, ON, 1, YES.
The following values can be used to specify FALSE: FALSE, OFF, 0, NO.

When setting extended options, use the following syntax:

```
CREATE SYNCHRONIZATION PROFILE myprofile 's=mysub;e={ctp=tcpip;adr=''host=localhost;port=2439''}'
```

**Privileges**

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

**Side effects**

Automatic commit.

**See also**

- “ALTER SYNCHRONIZATION PROFILE statement [MobiLink]” on page 481
- “DROP SYNCHRONIZATION PROFILE statement [MobiLink]” on page 778
- “SYNCHRONIZE statement [MobiLink]” on page 1011

**Standards and compatibility**

- SQL/2008  Vendor extension.

---

**CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]**

Creates a subscription in a SQL Anywhere remote database between a MobiLink user and a publication.

**Syntax**

```
CREATE SYNCHRONIZATION SUBSCRIPTION [ subscription-name ]
TO publication-name
[ FOR ml-username, ... ]
[ TYPE network-protocol ]
[ ADDRESS protocol-options ]
[ OPTION option=value, ... ]
[ SCRIPT VERSION script-version ]
```

```
subscription-name : identifier
ml-username : identifier
network-protocol : http | https | tls | tcpip
protocol-options : string
value : string | integer
script-version : string
```
Parameters

subscription-name A unique name that you can use to identify this subscription. It is strongly recommended that you name all your subscriptions.

TO clause This clause specifies the name of a publication.

FOR clause This clause specifies one or more MobiLink user names. If you specify more than one user name, a separate subscription is created for each user. If you specify a subscription name, only one MobiLink user name can be specified.

ml-username is a user who is authorized to synchronize with the MobiLink server.

Omit the FOR clause to set the protocol type, protocol options, and extended options for a publication. If the FOR clause is omitted, you cannot specify a subscription name or use the SCRIPT VERSION clause.

TYPE clause This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause This clause specifies network protocol options such as the location of the MobiLink server. Multiple options must be separated with semicolons.

OPTION clause This clause allows you to set extended options for the subscription. If no FOR clause is provided, the extended options act as default settings for the publication.

SCRIPT VERSION clause This clause specifies the script version to use during synchronization. Typically, you must specify a new script version for each schema change you implement.

You cannot use the SCRIPT VERSION clause if the FOR clause is omitted.

Remarks

If no subscription-name is specified, a unique name is generated. The generated subscription name is the same as the publication name, provided it is unique. Otherwise, a unique name is formed by adding a number to the end of the publication name, for example, pub001, pub002, and so on.

The network-protocol, protocol-options, and option values can be set in several places.

This statement causes options and other information to be stored in the SQL Anywhere ISYSSYNC system table. Anyone with privileges to view data in the SYSSYNC system view can view the information, which could include passwords and encryption certificates. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

You must have exclusive access to all tables referenced in the publication to execute the statement.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.
Standards and compatibility

- SQL/2008  Vendor extension.

Examples

The following example creates a subscription named sales between the MobiLink user SSinger and the publication called sales_publication owned by user. When the subscription is synchronized, the script version sales_v1 is used and tables are locked in exclusive mode:

```
CREATE SYNCHRONIZATION SUBSCRIPTION sales
TO user.sales_publication
FOR SSinger
OPTION locktables='exclusive'
SCRIPT VERSION 'sales_v1'
```

The following example omits the FOR clause and stores settings for the publication called sales_publication:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
TO user.sales_publication
ADDRESS 'host=test.internal;port=2439;
OPTION locktables=exclusive';
```

CREATE SYNCHRONIZATION USER statement [MobiLink]

Creates a MobiLink user in a SQL Anywhere remote database.

Syntax

```
CREATE SYNCHRONIZATION USER ml-username
[ TYPE network-protocol ]
[ ADDRESS protocol-options ]
[ OPTION option=value, ... ]
```

ml-username : identifier
network-protocol:
tcpip
  | http
  | https
  | tls

protocol-options : string

value : string | integer

Parameters

ml_username A name identifying a MobiLink user.

TYPE clause This clause specifies the network protocol to use for synchronization. The default protocol is tcpip.

ADDRESS clause This clause specifies protocol-options in the form keyword=value, separated by semicolons. Which settings you supply depends on the communication protocol you are using (TCP/IP, TLS, HTTP, or HTTPS).

OPTION clause The OPTION clause allows you to set extended options using option=value in a comma-separated list.

The values for each option cannot contain equal signs or semicolons. The database server accepts any option that you enter without checking for its validity. Therefore, if you misspell an option or enter an invalid value, no error message appears until you run the dbmlsync command to perform synchronization.

Options set for a synchronization user can be overridden in individual subscriptions or on the dbmlsync command line.

The network-protocol, protocol-options, and options can be set in several places.

This statement causes options and other information to be stored in the SQL Anywhere ISYSSYNC system table. Anyone with the correct privileges to view the SYSSYNC system view can view the information, which could include passwords and encryption certificates. To avoid this potential security issue, you can specify the information on the dbmlsync command line.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.
See also

- “ALTER SYNCHRONIZATION USER statement [MobiLink]” on page 485
- “DROP SYNCHRONIZATION USER statement [MobiLink]” on page 779
- “CommunicationType (ctp) extended option” [MobiLink - Client Administration]
- “MobiLink client/server communications encryption” [SQL Anywhere Server - Database Administration]
- “SYSSYNC system view” on page 1396
- “dbmlsync syntax” [MobiLink - Client Administration]
- “MobiLink users” [MobiLink - Client Administration]
- “MobiLink client network protocol options” [MobiLink - Client Administration]
- “MobiLink SQL Anywhere client extended options” [MobiLink - Client Administration]
- “Priority order” [MobiLink - Client Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Examples

The following example creates a MobiLink user named SSinger, who synchronizes over TCP/IP with a server computer named mlserver.mycompany.com using the password Sam. The use of a password in the user definition is not secure.

```sql
CREATE SYNCHRONIZATION USER SSinger
    TYPE http
    ADDRESS 'host=mlserver.mycompany.com'
    OPTION MobiLinkPwd='Sam';
```

CREATE TABLE statement

Creates a new table in the database and, optionally, to create a table on a remote server.

Syntax

```
CREATE [GLOBAL TEMPORARY] TABLE [IF NOT EXISTS] [owner.]table-name
    ( { column-definition | table-constraint | pctfree }, ... )
    [ { IN | ON } dbspace-name ]
    [ ENCRYPTED ]
    [ ON COMMIT { DELETE | PRESERVE } ROWS ]
    [ NOT TRANSACTIONAL ]
    [ AT location-string ]
    [ SHARE BY ALL ]
```

```
column-definition:
column-name data-type
    [ COMPRESSED ]
    [ INLINE { inline-length | USE DEFAULT } ]
    [ PREFIX { prefix-length | USE DEFAULT } ]
    [ [ NO ] INDEX ]
    [ [ NOT | NULL ]
    [ DEFAULT default-value | IDENTIFY ]
    [ column-constraint ... ]
```
default-value:
special-value
| string
| global-variable
| [ - ] number
| ( constant-expression )
| ( sequence-expression )
| built-in-function( constant-expression )
| AUTOINCREMENT
| GLOBAL AUTOINCREMENT [ ( partition-size ) ]

special-value:
| CURRENT DATABASE
| CURRENT DATE
| CURRENT TIME
| [ CURRENT ] TIMESTAMP
| CURRENT PUBLISHER
| CURRENT REMOTE USER
| [ CURRENT ] USER
| [ CURRENT ] UTC TIMESTAMP
| LAST USER
| NULL

column-constraint:
[ CONSTRAINT constraint-name ] {
| UNIQUE [ CLUSTERED ]
| PRIMARY KEY [ CLUSTERED ] [ ASC | DESC ]
| REFERENCES table-name [ ( column-name ) ]
| MATCH [ UNIQUE ] { SIMPLE | FULL } ]
| action-list ] [ CLUSTERED ]
| CHECK ( condition )
| COMPUTE ( expression )
}

table-constraint:
[ CONSTRAINT constraint-name ] {
| UNIQUE [ CLUSTERED ] ( column-name [ ASC | DESC ], ... )
| PRIMARY KEY [ CLUSTERED ] ( column-name [ ASC | DESC ], ... )
| CHECK ( condition )
| foreign-key-constraint
}

foreign-key-constraint:
[ NOT NULL ] FOREIGN KEY [ role-name ]
| ( column-name [ ASC | DESC ], ... )
| REFERENCES table-name
| [ ( column-name, ... ) ]
| MATCH [ UNIQUE ] { SIMPLE | FULL } ]
| action-list ] [ CHECK ON COMMIT ] [ CLUSTERED ] [ FOR OLAP WORKLOAD ]

action-list:
[ ON UPDATE action ]
[ ON DELETE action ]

action:
CASCADE
| SET NULL
| SET DEFAULT | RESTRICT |

location-string : 
remote-server-name.[db-name].[owner].object-name |
remote-server-name[db-name][owner].object-name

pctfree : PCTFREE percent-free-space

percent-free-space : integer

Parameters

IN clause  Use this clause to specify the dbspace in which the base table is located. If this clause is not specified, then the base table is created in the dbspace specified by the default_dbspace option.

Temporary tables can only be created in the temporary dbspace. If you are creating a GLOBAL TEMPORARY table, and specify IN, the table is created in the temporary dbspace. If you specify a user-defined dbspace, an error is returned.

ENCRYPTED clause  The encrypted clause specifies that the table should be encrypted. You must enable table encryption when you create a database if you plan to encrypt tables. The table is encrypted using the encryption key and algorithm specified at database creation time.

ON COMMIT clause  The ON COMMIT clause is allowed only for temporary tables. By default, the rows of a temporary table are deleted on COMMIT. If the SHARE BY ALL clause is specified, either ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL must be specified.

NOT TRANSACTIONAL clause  The NOT TRANSACTIONAL clause is allowed when creating a global temporary table. A table created using NOT TRANSACTIONAL is not affected by either COMMIT or ROLLBACK. If the SHARE BY ALL clause is specified, either ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL must be specified.

AT clause  Create a remote table on a different server specified by location-string, and a proxy table on the current database that maps to the remote table. The AT clause supports the semicolon (;) as a field delimiter in location-string. If no semicolon is present, a period is the field delimiter. This syntax allows file names and extensions to be used in the database and owner fields.

For example, the following statement maps the table proxy_Customers to a new table Customers in a fictitious Microsoft Access database called MyAccessDB:

```
CREATE TABLE proxy_Customers
  (
    ID            INTEGER CONSTRAINT PRIMARY KEY,
    ClientName    TEXT,
    ClientAddress TEXT,
    Telephone     TEXT
  )
AT 'MyAccessDB;;Customers';
```

The string in the AT clause can also contain local or global variable names enclosed in braces ({variable-name}). The SQL variable name must be of type CHAR, VARCHAR, or LONG VARCHAR. For example, an AT clause that contains 'access;{@myfile};;a1' indicates that @myfile is a SQL...
variable and that the current contents of the @myfile variable should be substituted when the proxy table is created.

Windows Mobile does not support the AT clause.

Foreign key definitions are ignored on remote tables. Foreign key definitions on local tables that refer to remote tables are also ignored. Primary key definitions are sent to the remote server if the database server supports primary keys.

**SHARE BY ALL clause**  Use this clause only when creating global temporary tables to allow the table to be shared by all connections to the database. If the SHARE BY ALL clause is specified, either ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL must be specified.

**IF NOT EXISTS clause**  No changes are made if the named table already exists, and an error is not returned.

**column-definition**  Define a column in the table. The following are part of column definitions.

- **column-name**  The column name is an identifier. Two columns in the same table cannot have the same name.

- **data-type**  The type of data stored in the column.

- **COMPRESSED clause**  Compress the column.

**INLINE and PREFIX clauses**  The INLINE clause specifies the maximum BLOB size, in bytes, to store within the row. BLOBs smaller than or equal to the value specified by the INLINE clause are stored within the row. BLOBs that exceed the value specified by the INLINE clause are stored outside the row in table extension pages. Also, a copy of some bytes from the beginning of the BLOB may be kept in the row when a BLOB is larger than the INLINE value. Use the PREFIX clause to specify how many bytes are kept in the row. The PREFIX clause can improve the performance of requests that need the prefix bytes of a BLOB to determine if a row is accepted or rejected.

The prefix data for a compressed column is stored uncompressed, so if all the data required to satisfy a request is stored in the prefix, no decompression is necessary.

If neither INLINE nor PREFIX is specified, or if USE DEFAULT is specified, default values are applied as follows:

- For character data type columns, such as CHAR, NCHAR, and LONG VARCHAR, the default value of INLINE is 256, and the default value of PREFIX is 8.

- For binary data type columns, such as BINARY, LONG BINARY, VARBINARY, BIT, VARBIT, LONG VARBIT, BIT VARYING, and UUID, the default value of INLINE is 256, and the default value of PREFIX is 0.
Note
It is strongly recommended that you use the default values unless there are specific circumstances that require a different setting. The default values have been chosen to balance performance and disk space requirements. For example, if you set INLINE to a large value, and all the BLOBs are stored inline, row processing performance may degrade. If you set PREFIX too high, you increase the amount of disk space required to store BLOBs since the prefix data is a duplicate of a portion of the BLOB.

If only one of the values is specified, the other value is automatically set to the largest amount that does not conflict with the specified value. Neither the INLINE nor PREFIX value can exceed the database page size. Also, there is a small amount of overhead reserved in a table page that cannot be used to store row data. Therefore, specifying an INLINE value approximate to the database page size can result in a slightly smaller number of bytes being stored inline.

INDEX and NO INDEX clauses When storing BLOBs (character or binary types only), specify INDEX to create BLOB indexes on inserted values that exceed the internal BLOB size threshold (approximately eight database pages). This is the default behavior.

BLOB indexes can improve performance when random access searches within the BLOBs are required. However, for some types of BLOB values, such as images and multimedia files that will never require random-access, performance can improve if BLOB indexing is turned off. To turn off BLOB indexing for a column, specify NO INDEX.

Note
A BLOB index is not the same as a table index. A table index is created to index values in one or more columns.

NULL and NOT NULL clauses If NULL is specified, NULL values are allowed in the column. This is the default behavior. This setting is controlled by the allow_nulls_by_default database option.

By default, columns declared as BIT are not nullable, but they can be explicitly made nullable.

If NOT NULL is specified, NULL values are not allowed.

If the column is part of a UNIQUE or PRIMARY KEY constraint, the column cannot contain NULL, even if NULL is specified.

DEFAULT clause If a DEFAULT value is specified, it is used as the value for the column in any INSERT statement that does not specify a value for the column. If no DEFAULT value is specified, it is equivalent to DEFAULT NULL.

Following is a list of possible values for DEFAULT:

- special-value You use one of several specials values in the DEFAULT clause.

- [ CURRENT ] TIMESTAMP Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT TIMESTAMP, a default value is provided for inserts, and the value is updated with the current date and time of day whenever the row is updated.
To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT TIMESTAMP instead of DEFAULT TIMESTAMP.

Columns declared with DEFAULT TIMESTAMP contain unique values, so that applications can detect near-simultaneous updates to the same row. If the current TIMESTAMP value is the same as the last value, it is incremented by the value of the default_timestamp_increment option.

You can automatically truncate TIMESTAMP values in SQL Anywhere based on the default_timestamp_increment option. This is useful for maintaining compatibility with other database software that records less precise timestamp values.

The global variable @@dbts returns a TIMESTAMP value representing the last value generated for a column using DEFAULT TIMESTAMP.

- [ CURRENT ] UTC TIMESTAMP  Provides a way of indicating when each row in the table was last modified. When a column is declared with DEFAULT UTC TIMESTAMP, a default value is provided for inserts, and the value is updated with the current Coordinated Universal Time (UTC) whenever the row is updated.

To provide a default value on insert, but not update the column whenever the row is updated, use DEFAULT CURRENT UTC TIMESTAMP instead of DEFAULT UTC TIMESTAMP.

The behavior of this default is the same as TIMESTAMP and CURRENT TIMESTAMP except that the date and time of day is in Coordinated Universal Time (UTC).

- string  See “Strings” on page 6.
- global-variable  See “Global variables” on page 82.
- constant-expression  Constant expressions that do not reference database objects are allowed in a DEFAULT clause, so functions such as GETDATE or DATEADD can be used. If the expression is not a function or simple value, it must be enclosed in parentheses.
- sequence-expression  You can set DEFAULT to the current value or next value from a sequence in the database.
- AUTOINCREMENT  When using AUTOINCREMENT, the column must be one of the integer data types, or an exact numeric type.

On inserts into the table, if a value is not specified for the AUTOINCREMENT column, a unique value larger than any other value in the column is generated. If an INSERT specifies a value for the column that is larger than the current maximum value for the column, that value is inserted and then used as a starting point for subsequent inserts.

Deleting rows does not decrement the AUTOINCREMENT counter. Gaps created by deleting rows can only be filled by explicit assignment when using an insert. After an explicit insert of a column value less than the maximum, subsequent rows without explicit assignment are still automatically incremented with a value of one greater than the previous maximum.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.
AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the max_identity column in the SYSTABCOL system view. When the next value to be generated exceeds the maximum value that can be stored in the column to which the AUTOINCREMENT is assigned, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated.

The next value to use for a column can be reset using the sa_reset_identity procedure.

○ **GLOBAL AUTOINCREMENT** This default is intended for use when multiple databases are used in a MobiLink synchronization environment or SQL Remote replication.

This option is similar to AUTOINCREMENT, except that the domain is partitioned. Each partition contains the same number of values. You assign each copy of the database a unique global database identification number. SQL Anywhere supplies default values in a database only from the partition uniquely identified by that database's number.

The partition size can be specified in parentheses immediately following the AUTOINCREMENT keyword. The partition size can be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, be exhausted.

If the column is of type BIGINT or UNSIGNED BIGINT, the default partition size is $2^{32} = 4294967296$; for columns of all other types, the default partition size is $2^{16} = 65536$. Since these defaults may be inappropriate, especially if your column is not of type INT or BIGINT, it is best to specify the partition size explicitly.

When using this default, the value of the public option global_database_id in each database must be set to a unique, non-negative integer. This value uniquely identifies the database and indicates from which partition default values are to be assigned. The range of allowed values is $np + 1$ to $p(n + 1)$, where $n$ is the value of the public option global_database_id and $p$ is the partition size. For example, if you define the partition size to be 1000 and set global_database_id to 3, then the range is from 3001 to 4000.

If the previous value is less than $p(n + 1)$, the next default value is one greater than the previous largest value in the column. If the column contains no values, the first default value is $np + 1$. Default column values are not affected by values in the column outside the current partition; that is, by numbers less than $np + 1$ or greater than $p(n + 1)$. Such values may be present if they have been replicated from another database via MobiLink or SQL Remote.

You can find the most recently inserted value of the column by inspecting the @@identity global variable.

GLOBAL AUTOINCREMENT values are maintained as signed 64-bit integers, corresponding to the data type of the max_identity column in the SYSTABCOL system view. When the supply of values within the partition has been exhausted, NULL is returned. If the column has been declared to not allow NULLs, as is true for primary key columns, a SQL error is generated. In this case, a new value of global_database_id should be assigned to the database to allow default values to be chosen from another partition. To detect that the supply of unused values is low and handle this condition, create an event of type GlobalAutoincrement.
Because the public option global_database_id cannot be set to a negative value, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

If the public option global_database_id is set to the default value of 2147483647, a NULL value is inserted into the column. If NULL values are not permitted, attempting to insert the row causes an error.

The next value to use for a column can be reset using the `sa_reset_identity` procedure.

- **LAST USER**  
  LAST USER is the user ID of the user who last modified the row.

  LAST USER can be used as a default value in columns with character data types.

  On INSERT, this default has the same effect as CURRENT USER.

  On UPDATE, if a column with a default of LAST USER is not explicitly modified, it is changed to the name of the current user.

  When used with DEFAULT TIMESTAMP or DEFAULT UTC TIMESTAMP, a default of LAST USER can be used to record (in separate columns) both the user and the date and time a row was last changed.

**IDENTITY clause**  
IDENTITY is a Transact-SQL-compatible alternative to using DEFAULT AUTOINCREMENT. In SQL Anywhere, a column defined with IDENTITY is implemented as DEFAULT AUTOINCREMENT.

**column-constraint and table-constraint clauses**  
Column and table constraints help ensure the integrity of data in the database. If a statement would cause a violation of a constraint, execution of the statement does not complete, any changes made by the statement before error detection are undone, and an error is reported. There are two classes of constraints that can be created: **check constraint**, and **referential integrity (RI) constraints**. Check constraints are used to specify conditions that must be satisfied by values of columns being put into the database. RI constraints establish a relationship between data in different tables that must be maintained in addition to specifying uniqueness requirements for data.

There are three types of RI constraints: primary key, foreign key, and unique constraint. When you create an RI constraint (primary key, foreign key or unique constraint), the database server enforces the constraint by implicitly creating an index on the columns that make up the key of the constraint. The index is created on the key for the constraint as specified. A key consists of an ordered list of columns and a sequencing of values (ASC/DESC) for each column.

Constraints can be specified on columns or tables. A column constraint refers to one column in a table, while a table constraint can refer to one or more columns in a table.

- **PRIMARY KEY clause**  
  A primary key uniquely defines each row in the table. Primary keys comprise one or more columns. A table cannot have more than one primary key. In a column-constraint clause, specifying PRIMARY KEY indicates that the column is the primary key for the table. In a table-constraint, you use the PRIMARY KEY clause to specify one or more columns that, when combined in the specified order, make up the primary key for the table.
The ordering of columns in a primary key need not match the respective ordinal numbers of the columns. That is, the columns in a primary key need not have the same physical order in the row. Additionally, you cannot specify duplicate column names.

When you create a primary key, an index for the key is automatically created. You can specify the sequencing of values in the index by specifying ASC (ascending) or DESC (descending) for each column. You can also specify whether to cluster the index, using the CLUSTERED keyword.

Columns included in primary keys cannot allow NULL. Each row in the table has a unique primary key value.

It is recommended that you do not use approximate data types such as FLOAT and DOUBLE for primary keys. Approximate numeric data types are subject to rounding errors after arithmetic operations.

Spatial columns cannot be included in a primary key.

- **FOREIGN KEY clause** A foreign key restricts the values for a set of columns to match the values in a primary key or a unique constraint of another table (the primary table). For example, a foreign key constraint could be used to ensure that a customer number in an invoice table corresponds to a customer number in the Customers table.

The foreign key column order does not need to reflect the order of columns in the table.

Duplicate column names are not allowed in the foreign key specification.

The default action is RESTRICT if no action is specified for an UPDATE or DELETE operation.

When you create a foreign key, an index for the key is automatically created. You can specify the sequencing of values in the index by specifying ASC (ascending) or DESC (descending) for each column. You can also specify whether to cluster the index, using the CLUSTERED keyword.

A global temporary table cannot have a foreign key that references a base table and a base table cannot have a foreign key that references a global temporary table.

If you attempt to add a foreign key to a column that does not exist, that column is automatically created.

- **NOT NULL clause** Disallow NULLs in the foreign key columns. A NULL in a foreign key means that no row in the primary table corresponds to this row in the foreign table.

- **role-name clause** The role name is the name of the foreign key. The main function of the role name is to distinguish between two foreign keys to the same table. If no role name is specified, the role name is assigned as follows:

  1. If there is no foreign key with a role name the same as the table name, the table name is assigned as the role name.

  2. If the table name is already taken, the role name is the table name concatenated with a zero-padded three-digit number unique to the table.
- **REFERENCES clause**  A foreign key constraint can be implemented using a REFERENCES column constraint (single column only) or a FOREIGN KEY table constraint, in which case the constraint can specify one or more columns. If you specify column-name in a REFERENCES column constraint, it must be a column in the primary table, must be subject to a unique constraint or primary key constraint, and that constraint must consist of only that one column. If you do not specify column-name, the foreign key column references the single primary key column of the primary table.

- **MATCH clause**  The MATCH clause determines what is considered a match when using a multi-column foreign key by allowing you to regulate what constitutes an orphaned row versus what constitutes a violation of referential integrity. The MATCH clause also allows you to specify uniqueness for the key, thereby eliminating the need to declare uniqueness separately.

The following is a list of MATCH types you can specify. See the Examples section at the end of this topic for a description of how the MATCH type affects matching behavior.

- **MATCH [ UNIQUE ] SIMPLE**  A match occurs for a row in the foreign key table if all the column values match the corresponding column values present in a row of the primary key table. A row is orphaned in the foreign key table if at least one column value in the foreign key is NULL.

  MATCH SIMPLE is the default behavior.

  If the UNIQUE keyword is specified, the referencing table can have only one match for non-NULL key values.

- **MATCH [ UNIQUE ] FULL**  A match occurs for a row in the foreign key table if none of the values are NULL and the values match the corresponding column values in a row of the primary key table. A row is orphaned if all column values in the foreign key are NULL.

  If the UNIQUE keyword is specified, the referencing table can have only one match for non-NULL key values.

- **UNIQUE clause**  In a column-constraint clause, a UNIQUE constraint specifies that the values in the column must be unique. In a table-constraint clause, the UNIQUE constraint identifies one or more columns that uniquely identify each row in the table. No two rows in the table can have the same values in all the named column(s). A table can have more than one UNIQUE constraint.

  A UNIQUE constraint is not the same as a unique index. Columns of a unique index are allowed to be NULL, while columns in a UNIQUE constraint are not. Also, a foreign key can reference either a primary key or a UNIQUE constraint, but cannot reference a unique index since a unique index can include multiple instances of NULL.

  Columns in a UNIQUE constraint can be specified in any order. Additionally, you can specify the sequencing of values in the corresponding index that is automatically created, by specifying ASC (ascending) or DESC (descending) for each column. You cannot specify duplicate column names, however.
It is recommended that you do not use approximate data types such as FLOAT and DOUBLE for columns with unique constraints. Approximate numeric data types are subject to rounding errors after arithmetic operations.

You can also specify whether to cluster the constraint, using the CLUSTERED keyword.

- **CHECK clause**  This constraint allows arbitrary conditions to be verified. For example, a CHECK constraint could be used to ensure that a column called Sex only contains the values M or F.

  If you need to create a CHECK constraint that involves a relationship between two or more columns in the table (for example, column A must be less than column B), define a table constraint instead.

  No row in a table is allowed to violate a CHECK constraint. If an INSERT or UPDATE statement would cause a row to violate the constraint, the operation is not permitted and the effects of the statement are undone. The change is rejected only if a CHECK constraint condition evaluates to FALSE, and the change is allowed if a CHECK constraint condition evaluates to TRUE or UNKNOWN.

- **COMPUTE clause**  The COMPUTE clause is only for use in a column-constraint clause.

  When a column is created using a COMPUTE clause, its value in any row is the value of the supplied expression. Columns created with this constraint are read-only columns for applications: the value is changed by the database server whenever the row is modified. The COMPUTE expression should not return a non-deterministic value. For example, it should not include a special value such as CURRENT_TIMESTAMP, or a non-deterministic function. If a COMPUTE expression returns a non-deterministic value, then it cannot be used to match an expression in a query.

  Any UPDATE statement that attempts to change the value of a computed column fires any triggers associated with the column.

- **CHECK ON COMMIT clause**  The CHECK ON COMMIT option overrides the wait_for_commit database option, and causes the database server to wait for a COMMIT before checking RESTRICT actions on a foreign key. The CHECK ON COMMIT option delays foreign key checking, but does not delay other actions such as CASCADE, SET NULL, SET DEFAULT, or check constraints.

- **FOR OLAP WORKLOAD clause**  When you specify FOR OLAP WORKLOAD in the REFERENCES clause of a foreign key definition, the database server performs certain optimizations and gathers statistics on the key to help improve performance for OLAP workloads, particularly when the optimization_workload option is set to OLAP.

- **PCTFREE clause**  Specifies the percentage of free space you want to reserve for each table page. The free space is used if rows increase in size when the data is updated. If there is no free space in a table page, every increase in the size of a row on that page requires the row to be split across multiple table pages, causing row fragmentation and possible performance degradation.

  The value \textit{percent-free-space} is an integer between 0 and 100. The former value specifies that no free space is to be left on each page—each page is to be fully packed. A high value causes each row to be inserted into a page by itself. If PCTFREE is not set, or is later dropped, the default PCTFREE value is
applied according to the database page size (200 bytes for a 4 KB (and up) page size). The value for PCTFREE is stored in the ISYSTAB system table.

Remarks

The CREATE TABLE statement creates a new table. A table can be created for another user by specifying an owner name. If GLOBAL TEMPORARY is specified, the table is a temporary table. Otherwise, the table is a base table.

Tables created by preceding the table name in a CREATE TABLE statement with a pound sign (#) are declared temporary tables, which are available only in the current connection. Temporary tables created with the pound sign (#) are identical to those created with the ON COMMIT PRESERVE ROWS clause.

Two local temporary tables within the same scope cannot have the same name. If you create a temporary table with the same name as a base table, the base table only becomes visible within the connection once the scope of the local temporary table ends. A connection cannot create a base table with the same name as an existing temporary table.

Columns in SQL Anywhere allow NULLs by default. This setting can be controlled using the allow_nulls_by_default database option.

Privileges

You must have the CREATE TABLE system privilege to create tables owned by you. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create tables owned by others.

To create proxy tables owned by you, you must have the CREATE PROXY TABLE system privilege. You must have the CREATE ANY TABLE or CREATE ANY OBJECT system privilege to create proxy tables owned by others.

Side effects

Automatic commit (even when creating global temporary tables).
Standards and compatibility

- **SQL/2008**  
  CREATE TABLE is a core feature of the SQL/2008 standard, though some of its components supported in SQL Anywhere are optional SQL language features. A subset of these features include:

  ○ Temporary table support is SQL language feature F531.

  ○ Support for IDENTITY columns is SQL feature T174, though SQL Anywhere uses slightly different syntax from that in the standard.

  ○ Foreign key constraint support includes SQL language features T191 "Referential action: RESTRICT", F741 "Referential MATCH types", F191 "Referential delete actions", and F701 "Referential update actions". SQL Anywhere does not support MATCH PARTIAL.

  SQL Anywhere does not support SQL language feature T591 ("UNIQUE constraints of possibly null columns"). In SQL Anywhere, all columns that are part of a PRIMARY KEY or UNIQUE constraint must be declared NOT NULL.

  The following components of CREATE TABLE are vendor extensions:
○ The \{ IN | ON \} \textit{dbspace-name} clause.

○ The ENCRYPTED, NOT TRANSACTIONAL, and SHARE BY ALL clauses.

○ The COMPRESSED, INLINE, PREFIX, and NO INDEX clauses of a column definition.

○ Various implementation-defined DEFAULT values, including AUTOINCREMENT, GLOBAL AUTOINCREMENT, CURRENT DATABASE, CURRENT REMOTE USER, CURRENT UTC TIMESTAMP, and most special values. A DEFAULT clause that references a SEQUENCE generator is also a vendor extension.

○ The specification of MATCH UNIQUE.

○ Sortedness specification (ASC or DESC) on a PRIMARY KEY or FOREIGN KEY clause.

○ The ability to specify FOREIGN KEY columns in an order different from that specified in the referenced table's PRIMARY KEY clause.

**Examples**

The following statement creates a table \textit{file_table} with two columns: \textit{file_name} and \textit{file_contents}. The contents column is LONG BINARY and is compressed:

```
CREATE TABLE file_table (  
  file_name VARCHAR(255),  
  file_contents LONG BINARY COMPRESSED  
);
```

The following example creates a table for a library database to hold book information.

```
CREATE TABLE library_books (  
  -- NOT NULL is assumed for primary key columns  
  isbn CHAR(20) PRIMARY KEY,  
  copyright_date DATE,  
  title CHAR(100),  
  author CHAR(50),  
  -- column(s) corresponding to primary key of room  
  -- are created automatically  
);
```

The following example creates a table for a library database to hold information about borrowed books. The default value for \textit{date_borrowed} indicates that the book is borrowed on the day the entry is made. The \textit{date_returned} column is NULL until the book is returned.

```
CREATE TABLE borrowed_book (  
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,  
  date_returned DATE,  
  book CHAR(20)  
  REFERENCES library_books (isbn),  
  -- The check condition is UNKNOWN until  
  -- the book is returned, which is allowed  
  CHECK( date_returned >= date_borrowed )  
);
```

The following example creates tables for a sales database to hold order and order item information.

```
CREATE TABLE Orders (  
  order_num INTEGER NOT NULL PRIMARY KEY,  
```
CREATE TABLE Order_item (  
  order_num INTEGER NOT NULL,  
  item_num SMALLINT NOT NULL,  
  PRIMARY KEY ( order_num, item_num ),  
  -- When an order is deleted, delete all of its  
  -- items,  
  FOREIGN KEY ( order_num )  
  REFERENCES Orders ( order_num )  
  ON DELETE CASCADE  
);  

The following example creates a table named t1 on a fictitious remote server, SERVER_A, and creates a proxy table named t1 that is mapped to the remote table.  

```
CREATE TABLE t1  
(  
  a INT,  
  b CHAR(10)  
)  
AT 'SERVER_A.db1.joe.t1';
```

The following example creates two tables named Table1 and Table2, adds foreign keys to Table2, and inserts values into Table1. The final statement attempts to insert values into Table2. An error is returned because the values that you attempt to insert are not a simple match with Table1.  

```
CREATE TABLE Table1 ( P1 INT, P2 INT, P3 INT, P4 INT, P5 INT, P6 INT,  
  PRIMARY KEY ( P1, P2 ) );  
CREATE TABLE Table2 ( F1 INT, F2 INT, F3 INT, PRIMARY KEY ( F1, F2 ) );  
ALTER TABLE Table2  
  ADD FOREIGN KEY fk2( F1,F2 )  
  REFERENCES Table1( P1, P2 )  
  MATCH SIMPLE;  
INSERT INTO Table1 (P1, P2, P3, P4, P5, P6) VALUES ( 1,2,3,4,5,6 );  
INSERT INTO Table2 (F1,F2) VALUES ( 3,4 );
```

The following statements show how MATCH SIMPLE and MATCH SIMPLE UNIQUE differ in how multi-column foreign keys are handled when some but not all of the columns in the key are NULL. This statement will fail because the values for the second column are not unique.  

```
CREATE TABLE pt( pk INT PRIMARY KEY, str VARCHAR(10));  
INSERT INTO pt VALUES(1,'one'), (2,'two');  
COMMIT;  
CREATE TABLE ft1( fpk INT PRIMARY KEY, FOREIGN KEY (ref) REFERENCES pt MATCH SIMPLE);  
INSERT INTO ft1 VALUES(100,1), (200,1); //This statement will insert 2 rows.  
CREATE TABLE ft2( fpk INT PRIMARY KEY, FOREIGN KEY (ref) REFERENCES pt MATCH UNIQUE SIMPLE);  
INSERT INTO ft2 VALUES(100,1), (200,1);
```

The following statements show how MATCH SIMPLE and MATCH UNIQUE SIMPLE differ:  

```
CREATE TABLE pt2(  
  pk1 INT NOT NULL,  
  pk2 INT NOT NULL,  
  str VARCHAR(10),  
  PRIMARY KEY (pk1,pk2));  
INSERT INTO pt2 VALUES(1,10,'one-ten'), (2,20,'two-twenty');
```
CREATE TABLE ft3(
    fpk INT PRIMARY KEY,
    ref1 INT,
    ref2 INT );

ALTER TABLE ft3 ADD FOREIGN KEY (ref1,ref2)
    REFERENCES pt2 (pk1,pk2) MATCH SIMPLE;

CREATE TABLE ft4(
    fpk INT PRIMARY KEY,
    ref1 INT,
    ref2 INT );

INSERT INTO ft3 VALUES(100,1,10);
// MATCH SIMPLE test succeeds; all column values match the corresponding
// values in pt2.
INSERT INTO ft3 VALUES(200,null,null);
// MATCH SIMPLE test succeeds; at least one column in the key is null.
INSERT INTO ft3 VALUES(300,2,null);
// MATCH SIMPLE test succeeds; at least one column in the key is null.
INSERT INTO ft4 VALUES(100,1,10);
// MATCH FULL test succeeds; all column values match the corresponding
// values in pt2.
INSERT INTO ft4 VALUES(200,null,null);
// MATCH FULL test succeeds; all column values in the key are null.
INSERT INTO ft4 VALUES(300,2,null);
// MATCH FULL test fails; both columns in the key must be null or match the
// corresponding values in pt2.

CREATE TEMPORARY TRACE EVENT statement

Creates a user trace event that persists until the database is stopped.

Syntax

CREATE [ OR REPLACE ] TEMPORARY TRACE EVENT trace-event-name
[ WITH SEVERITY { 0-255 |
    ALWAYS |
    CRITICAL |
    ERROR |
    WARNING |
    INFORMATION |
    DEBUG } ]
[ field-name field-type [, ... ] ]

Parameters

trace-event-name Specify a user trace event name. User trace event names cannot start with the prefix SYS_. System trace event names cannot be specified.

OR REPLACE clause Create a new trace event, or replace an existing trace event with the same name.
**WITH SEVERITY clause**  If the severity level is not specified, the default severity level (DEBUG) is used. User trace events are owned by the database that the user was connected to when the trace event was created. The supported severity values are:

<table>
<thead>
<tr>
<th>Level</th>
<th>Severity value range</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALWAYS</td>
<td>0</td>
</tr>
<tr>
<td>CRITICAL</td>
<td>1-50</td>
</tr>
<tr>
<td>ERROR</td>
<td>51-100</td>
</tr>
<tr>
<td>WARNING</td>
<td>101-150</td>
</tr>
<tr>
<td>INFORMATION</td>
<td>151-200</td>
</tr>
<tr>
<td>DEBUG</td>
<td>201-255</td>
</tr>
</tbody>
</table>

**field-name**  The field that gathers information of a specific type from the user trace event. The field must be a valid identifier.

**field-type**  You can use any data type that is supported for a column except an array type.

**Remarks**  
A trace event is stored in memory and is dropped when the database server stops if it has not been dropped explicitly.

**System privileges**  
You must have the MANAGE ANY TRACE SESSION system privilege.

**Side effects**  
None
See also

- “Event tracing” [SQL Anywhere Server - Database Administration]
- “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
- “ALTER TRACE EVENT SESSION statement” on page 506
- “DROP TRACE EVENT statement” on page 783
- “DROP TRACE EVENT SESSION statement” on page 784
- “NOTIFY TRACE EVENT statement” on page 903
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_fields system procedure” on page 1312
- “sp_trace_event_sessions system procedure” on page 1317
- “sp_trace_event_session_events system procedure” on page 1314
- “sp_trace_event_session_targets system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement creates a user trace event:

```sql
CREATE TEMPORARY TRACE EVENT my_event( id INTEGER, information LONG VARCHAR );
```

CREATE TEMPORARY TRACE EVENT SESSION statement

Creates a user trace event session.

Syntax

```sql
CREATE [ OR REPLACE ] TEMPORARY TRACE EVENT SESSION session-name
[ event-definition [ , ... ] ]
[ target-definition [ , ... ] ]
```

`event-definition`:

```sql
ADD TRACE EVENT event-name
```

`target-definition`:

```sql
ADD TARGET target-name
[ SET target-parameter-name=target-parameter-value [ ,... ] ]
```

`target-name`:

```sql
FILE
```

`target-parameter-name`:

```sql
{ filename_prefix | max_size | num_files }
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 707
Parameters

**session-name**  The name of the trace event session.

**event-name**  The name of the trace event to add to the session. System- and user-defined trace events are supported. Call the sp_trace_events system procedure to obtain a list of system-defined trace events.

**target-name**  The only supported value is FILE.

**target-parameter-name**  The following target parameters are supported:

<table>
<thead>
<tr>
<th>target-parameter-name</th>
<th>target-parameter-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename_prefix</td>
<td>An ETD file name prefix with or without a path. All ETD files have the extension .etd. This parameter is required.</td>
</tr>
<tr>
<td>max_size</td>
<td>The maximum size of the file in bytes. The default is 0, which means there is no limit on the file size and it grows as long as disk space is available. Once the specified size is reached, a new file is started.</td>
</tr>
<tr>
<td>num_files</td>
<td>The number of files where event tracing information is written, and it is used only if max_size is set. If all the files reach the maximum specified size, the database server starts overwriting the oldest file.</td>
</tr>
<tr>
<td>flush_on_write</td>
<td>A value that controls whether disk buffers are flushed for each event that is logged. The values yes, true, no, and false are accepted. The default is false. When this parameter is turned on, the performance of the database server may be reduced if many trace events are being logged.</td>
</tr>
<tr>
<td>compressed</td>
<td>A value that controls compression of the ETD file to conserve disk space. The values on and off are accepted. The default is off.</td>
</tr>
</tbody>
</table>

**OR REPLACE** clause  This clause creates a trace event session or replaces an existing trace event session with the same name.

Remarks

Trace event sessions do not run until they are explicitly started with the ALTER TRACE EVENT SESSION statement. Trace event sessions can be used to capture trace events related to system behavior or for a particular user. Trace event sessions are stored in memory and are dropped when the database server stops if they have not been dropped explicitly.

System privileges

You must have the MANAGE ANY TRACE SESSION system privilege.
Side effects

None

See also

- “Event tracing” [SQL Anywhere Server - Database Administration]
- “CREATE TEMPORARY TRACE EVENT statement” on page 705
- “ALTER TRACE EVENT SESSION statement” on page 506
- “DROP TRACE EVENT statement” on page 783
- “DROP TRACE EVENT SESSION statement” on page 784
- “NOTIFY TRACE EVENT statement” on page 903
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_fields system procedure” on page 1312
- “sp_trace_event_sessions system procedure” on page 1317
- “sp_trace_event_session_events system procedure” on page 1314
- “sp_trace_event_session_targets system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]
- “-sf database server option” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement creates an event tracing session that records information about the user-defined event my_event and the system-defined event ConsoleLogInformation to a file named my_trace_file:

```
CREATE TEMPORARY TRACE EVENT SESSION my_session
  ADD TRACE EVENT my_event, -- user event
  ADD TRACE EVENT SYS_ConsoleLog_Information -- system event
  ADD TARGET FILE ( SET filename_prefix='my_trace_file' ); -- add a target
```

CREATE TEXT CONFIGURATION statement

Creates a text configuration object for use with building and updating text indexes.

Syntax

```
CREATE TEXT CONFIGURATION [ owner. ] new-config-name
  FROM [ owner. ] existing-config-name
```

Parameters

- **FROM clause** Specify the name of a text configuration object to use as the template for creating the new one. The names of the default text configuration objects are default_char and default_nchar.
When you create a text configuration object, the database options that affect how date and time values are converted to strings are copied from the default_char and default_nchar text configuration object templates.

**Remarks**

You create a text configuration object using another text configuration object as a template and then alter the options as needed using the ALTER TEXT CONFIGURATION statement.

To view the list of all text configuration objects in the database, and their settings, query the SYSTEXTCONFIG system view.

**Privileges**

You must have the CREATE TEXT CONFIGURATION system privilege to create text configurations objects owned by you. You must have the CREATE ANY TEXT CONFIGURATION or CREATE ANY OBJECT system privilege to create text configuration objects owned by others.

All text configuration objects have PUBLIC access. Any user with privilege to create a text index can also use any text configuration object.

**Side effects**

Automatic commit

**See also**

- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Default text configuration objects” [SQL Anywhere Server - SQL Usage]
- “Database options that impact text configuration objects” [SQL Anywhere Server - SQL Usage]
- “What to specify when creating or altering text configuration objects” [SQL Anywhere Server - SQL Usage]
- “SYSTEXTCONFIG system view” on page 1405
- “Tutorial: Performing a full text search on a GENERIC text index” [SQL Anywhere Server - SQL Usage]
- “Tutorial: Performing a fuzzy full text search” [SQL Anywhere Server - SQL Usage]
- “ALTER TEXT CONFIGURATION statement” on page 501
- “DROP TEXT CONFIGURATION statement” on page 781
- “sa_refresh_text_indexes system procedure” on page 1211

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following CREATE TEXT CONFIGURATION statement creates a text configuration object, max_term_sixteen, using the default_char text configuration object. The subsequent ALTER TEXT CONFIGURATION statement changes the maximum term length for max_term_sixteen to 16.

```sql
CREATE TEXT CONFIGURATION max_term_sixteen FROM default_char;
ALTER TEXT CONFIGURATION max_term_sixteen
  MAXIMUM TERM LENGTH 16;
```
CREATE TEXT INDEX statement

Creates a text index.

Syntax

```
CREATE TEXT INDEX [ IF NOT EXISTS ] text-index-name
ON [ owner. ] { table-name | mv-name } { column-name, ... }
[ IN dbspace-name ]
[ CONFIGURATION [ owner. ]text-configuration-name ]
[ { IMMEDIATE REFRESH |
  MANUAL REFRESH |
  AUTO REFRESH [ EVERY integer { MINUTES | HOURS } ] } ]
```

Parameters

**IF NOT EXISTS clause**  When the IF NOT EXISTS clause is specified and the named text index exists, no changes are made and an error is not returned.

**ON clause**  Specify the table and columns on which to build the text index.

**IN clause**  Specify the dbspace in which the text index is located. If this clause is not specified, then the text index is created in the same dbspace as the table it references.

**CONFIGURATION clause**  Specify the text configuration object to use when creating the text index. If this clause is not specified, the default_nchar text configuration object is used if any of the columns in the index are NCHAR; otherwise, the default_char text configuration object is used.

**REFRESH clause**  Specify the refresh type for the text index. If you do not specify a REFRESH clause, IMMEDIATE REFRESH is used as the default. You can specify the following refresh types:

- **IMMEDIATE REFRESH**  Refreshes the text index each time changes in the underlying table or the materialized view impact data in the text index.

- **AUTO REFRESH**  Refreshes the text index automatically using an internal server event. Use the EVERY sub-clause to specify the refresh interval in minutes or hours. If you specify AUTO REFRESH without supplying interval information, the database server refreshes the text index every 60 minutes. A text index may be refreshed earlier than the interval specified by the AUTO REFRESH clause if the pending_size value, as returned by the sa_text_index_stats system procedure, exceeds 20% of the text index size at the last refresh or if the deleted_length exceeds 50% of the text index size. An internal event executes once per minute to check this condition for all AUTO REFRESH text indexes.

- **MANUAL REFRESH**  The text index is refreshed manually.

For more information about refresh types, see “Text index refresh types” [SQL Anywhere Server - SQL Usage].

Remarks

You cannot create a text index on a regular view or a temporary table.
Once a text index is created on a materialized view, it cannot be refreshed or truncated, it can only be dropped. The text index on a materialized view is maintained by the database server whenever the underlying materialized view is refreshed or updated.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

An IMMEDIATE REFRESH text index on a base table is populated at creation time and an exclusive lock is held on the table during this initial refresh. IMMEDIATE REFRESH text indexes provide full support for queries that use snapshot isolation.

An IMMEDIATE REFRESH text index on a materialized view is populated at creation time if the view is populated.

MANUAL and AUTO REFRESH text indexes must be initialized (refreshed) after creation.

Refreshes for AUTO REFRESH text indexes scan the table using isolation level 0.

Once a text index is created, you cannot change it to, or from, being defined as IMMEDIATE REFRESH. If either of these changes is required, drop and recreate the text index.

You can choose to manually refresh an AUTO REFRESH text index by using the REFRESH TEXT INDEX statement.

To view text indexes and the text configuration objects they refer to, see “Viewing text index terms and settings (Sybase Central)” [SQL Anywhere Server - SQL Usage].

Privileges
To create a text index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

To create a text index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- CREATE ANY INDEX system privilege
- CREATE ANY OBJECT system privilege

Side effects
Automatic commit
See also

- “Tutorial: Performing a full text search on a GENERIC text index” [SQL Anywhere Server - SQL Usage]
- “isolation_level option” [SQL Anywhere Server - Database Administration]
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]
- “REFRESH TEXT INDEX statement” on page 928
- “Tutorial: Performing a fuzzy full text search” [SQL Anywhere Server - SQL Usage]
- “Text index concepts and reference” [SQL Anywhere Server - SQL Usage]
- “SYSTEXTCONFIG system view” on page 1405
- “ALTER TEXT INDEX statement” on page 504
- “DROP TEXT INDEX statement” on page 782
- “REFRESH TEXT INDEX statement” on page 928
- “TRUNCATE TEXT INDEX statement” on page 1021
- “sa_char_terms system procedure” on page 1095
- “sa_nchar_terms system procedure” on page 1195
- “sa_refresh_text_indexes system procedure” on page 1211
- “sa_text_index_stats system procedure” on page 1256

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example creates a text index, myTxtIdx, on the Description column of the MarketingInformation table in the sample database. The MarketingTextConfig text configuration object is used, and the refresh interval is set to every 24 hours.

```
CREATE TEXT INDEX myTxtIdx ON GROUPO.MarketingInformation ( Description )
CONFIGURATION default_char
AUTO REFRESH EVERY 24 HOURS;
```

CREATE TRIGGER statement

Creates a trigger on a table.

Syntax

```
CREATE [ OR REPLACE ] TRIGGER trigger-name trigger-type
{ trigger-event-list | UPDATE OF column-list }
[ ORDER integer ] ON table-name
[ REFERENCING [ OLD AS old-name ]
 [ NEW AS new-name ]
 [ REMOTE AS remote-name ] ]
[ FOR EACH { ROW | STATEMENT } ]
[ WHEN ( search-condition ) ]
trigger-body
```

```
column-list : column-name[, ...]
```

```
trigger-type :
BEFORE
```
AFTER
INSTEAD OF
RESOLVE

\(\text{trigger-event-list} : \text{trigger-event[, ... ]}\)

\(\text{trigger-event} : \)
\(\text{DELETE}\)
\(\text{INSERT}\)
\(\text{UPDATE} (\text{column-name})\)
\(\text{UPDATING} [ (\text{column-name-string}) ]\)

\(\text{trigger-body} : \) a BEGIN statement. See “BEGIN statement” on page 523.

Parameters

**OR REPLACE clause** Specifying OR REPLACE creates a new trigger, or replaces an existing trigger with the same name.

**trigger-event** Triggers can be fired by the following events. You can define either multiple triggers for DELETE, INSERT, or UPDATE events, or one trigger for an UPDATE OF column-list event:

- **DELETE clause** Invoked whenever a row of the associated table is deleted.

- **INSERT clause** Invoked whenever a new row is inserted into the table associated with the trigger.

- **UPDATE clause** Invoked whenever a row of the associated table is updated.

  If you specify an UPDATE clause, you must also supply a REFERENCING clause to avoid syntax errors.

- **UPDATE OF column-list clause** Invoked whenever a row of the associated table is updated and a column in the column-list is modified. This type of trigger event cannot be used in a trigger-event-list; it must be the only trigger event defined for the trigger. This clause cannot be used in an INSTEAD OF trigger.

  You can write separate triggers for each event that you need to handle or, if you have some shared actions and some actions that depend on the event, you can create a trigger for all events and use an IF statement to distinguish the action taking place.

- **UPDATING clause** The argument for UPDATING is a quoted string (for example, \(\text{UPDATING} (\text{mycolumn})\)). The argument for UPDATE is an identifier (for example, \(\text{UPDATE} (\text{mycolumn})\)). The two versions are interoperable, and are included for compatibility with SQL dialects of other vendors’ DBMS.

  If you specify an UPDATING clause, then you must also supply a REFERENCING clause to avoid syntax errors.

**trigger-type** Row-level triggers can be defined to execute BEFORE, AFTER, or INSTEAD OF an insert, update, or delete operation. Statement-level triggers can be defined to execute INSTEAD OF or AFTER the statement.
BEFORE UPDATE triggers fire any time an UPDATE occurs on a row, whether the new value differs from the old value. That is, if a `column-list` is specified for a BEFORE UPDATE trigger, then the trigger fires if any of the columns in `column-list` appear in the SET clause of the UPDATE statement. If a `column-list` is specified for an AFTER UPDATE trigger, then the trigger is fired only if the value of any of the columns in `column-list` is changed by the UPDATE statement.

INSTEAD OF triggers are the only form of trigger that you can define on a regular view. INSTEAD OF triggers replace the triggering action with another action. When an INSTEAD OF trigger fires, the triggering action is skipped and the specified action is performed. INSTEAD OF triggers can be defined as a row-level or a statement-level trigger. A statement-level INSTEAD OF trigger replaces the entire statement, including all row-level operations. If a statement-level INSTEAD OF trigger fires, then no row-level triggers fire as a result of that statement. However, the body of the statement-level trigger could perform other operations that, in turn, cause other row-level triggers to fire.

If you are defining an INSTEAD OF trigger, then you cannot use the UPDATE OF `column-list` clause, the ORDER clause, or the WHEN clause.

The RESOLVE trigger type is for use with SQL Remote; it fires before row-level UPDATE or UPDATE OF `column-list` only.

**ORDER clause** When defining additional triggers of the same type (insert, update, or delete) to fire at the same time (before, after, or resolve), you must specify an ORDER clause to tell the database server the order in which to fire the triggers. Order numbers must be unique among same-type triggers configured to fire at the same time. If you specify an order number that is not unique, then an error is returned. Order numbers do not need to be in consecutive order (for example, you could specify 1, 12, 30). The database server fires the triggers starting with the lowest number.

If you omit the ORDER clause, or specify 0, then the database server assigns the order of 1. However, if another same-type trigger is already set to 1, then an error is returned.

When adding additional triggers, you may need to modify the existing same-type triggers for the event, depending on whether the actions of the triggers interact. If they do not interact, then the new trigger must have an ORDER value higher than the existing triggers. If they do interact, you need to consider what the other triggers do, and you may need to change the order in which they fire.

The ORDER clause is not supported for INSTEAD OF triggers since there can only be one INSTEAD OF trigger of each type (insert, update, or delete) defined on a table or view.

**REFERENCING clause** The REFERENCING OLD and REFERENCING NEW clauses allow you to refer to the inserted, deleted, or updated rows. With this clause an UPDATE is treated as a delete followed by an insert.

An INSERT takes the REFERENCING NEW clause, which represents the inserted row. There is no REFERENCING OLD clause.

A DELETE takes the REFERENCING OLD clause, which represents the deleted row. There is no REFERENCING NEW clause.

An UPDATE takes the REFERENCING OLD clause, which represents the row before the update, and it takes the REFERENCING NEW clause, which represents the row after the update.
The meanings of REFERENCING OLD and REFERENCING NEW differ, depending on whether the trigger is a row-level or a statement-level trigger. For row-level triggers, the REFERENCING OLD clause allows you to refer to the values in a row before an update or delete, and the REFERENCING NEW clause allows you to refer to the inserted or updated values. The OLD and NEW rows can be referenced in BEFORE and AFTER triggers. The REFERENCING NEW clause allows you to modify the new row in a BEFORE trigger before the insert or update operation takes place.

For statement-level triggers, the REFERENCING OLD and REFERENCING NEW clauses refer to declared temporary tables holding the old and new values of the rows.

The REFERENCING REMOTE clause is for use with SQL Remote. It allows you to refer to the values in the VERIFY clause of an UPDATE statement. It should be used only with RESOLVE UPDATE or RESOLVE UPDATE OF column-list triggers.

**FOR EACH clause**

To declare a trigger as a row-level trigger, use the FOR EACH ROW clause. To declare a trigger as a statement-level trigger, you can either use a FOR EACH STATEMENT clause or omit the FOR EACH clause. For clarity, it is recommended that you specify the FOR EACH STATEMENT clause if you are declaring a statement-level trigger.

**WHEN clause**

The trigger fires only for rows where the search-condition evaluates to true. The WHEN clause can be used only with row level triggers. This clause cannot be used in an INSTEAD OF trigger.

**trigger-body**

The trigger body contains the actions to take when the triggering action occurs, and consists of a BEGIN statement.

You can include trigger operation conditions in the BEGIN statement. Trigger operation conditions perform actions depending on the trigger event that caused the trigger to fire. For example, if the trigger is defined to fire for both updates and deletes, you can specify different actions for the two conditions.

**Remarks**

The CREATE TRIGGER statement creates a trigger associated with a table in the database, and stores the trigger in the database.

You cannot define a trigger on a materialized view. If you do, a SQLE_INVALID_TRIGGER_MATVIEW error is returned.

A trigger is declared as either a row-level trigger, in which case it executes before or after each row is modified, or a statement-level trigger, in which case it executes after the entire triggering statement is completed.

CREATE TRIGGER puts a table lock on the table and requires exclusive use of the table.
Privileges
You must have the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege. Additionally, you must be the owner of the table the trigger is built on or have one of the following privileges:

- ALTER privilege on the table
- ALTER ANY TABLE system privilege
- ALTER ANY OBJECT system privilege

To create a trigger on a view owned by someone else, you must have either the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege, and you must have either the ALTER ANY VIEW or ALTER ANY OBJECT system privilege.

Side effects
Automatic commit.

See also
- “BEGIN statement” on page 523
- “CREATE TRIGGER statement [T-SQL]” on page 720
- “DROP TRIGGER statement” on page 785
- “ROLLBACK TRIGGER statement” on page 952
- “UPDATE statement” on page 1037
- “ALTER TRIGGER statement” on page 507
- “RAISERROR statement” on page 920
- “CONFLICT function [Miscellaneous]” on page 182
- “INSTEAD OF triggers” [SQL Anywhere Server - SQL Usage]
- “SQL statements for implementing integrity constraints” [SQL Anywhere Server - SQL Usage]
- “Creating a trigger on a table (Sybase Central)” [SQL Anywhere Server - SQL Usage]
- “Triggers” [SQL Anywhere Server - SQL Usage]
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008 CREATE TRIGGER is part of optional SQL language feature T211 "Basic trigger capability" of the SQL/2008 standard. Row triggers are optional SQL language feature T212, while INSTEAD OF triggers are optional SQL language feature T213.

Some features of SQL Anywhere triggers are vendor extensions. These include:

- The optional OR REPLACE syntax. If an existing trigger is replaced, authorization of the creation of the new trigger instance is bypassed.

- The ORDER clause. In SQL/2008, triggers are fired in the order they were created.

- RESOLVE triggers are a vendor extension.

- Transact-SQL ROW and RESOLVE triggers are not supported by Adaptive Server Enterprise. The SQL Anywhere Transact-SQL dialect does not support Transact-SQL INSTEAD OF triggers,
though these are supported by Adaptive Server Enterprise. Transact-SQL triggers are defined using different syntax.

Example

This example creates a statement-level trigger. First, create a table as shown in this CREATE TABLE statement (requires the CREATE TABLE system privilege):

```
CREATE TABLE t0
(id INTEGER NOT NULL,
times TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
remarks TEXT NULL,
PRIMARY KEY (id))
```

Next, create a statement-level trigger for this table:

```
CREATE TRIGGER myTrig AFTER INSERT ORDER 4 ON t0 REFERENCING NEW AS new_name FOR EACH STATEMENT
BEGIN
  DECLARE @id1 INTEGER;
  DECLARE @times1 TIMESTAMP;
  DECLARE @remarks1 LONG VARCHAR;
  DECLARE @err_notfound EXCEPTION FOR SQLSTATE VALUE '02000';
  //declare a cursor for table new_name
  DECLARE new1 CURSOR FOR
  SELECT id, times, remarks FROM new_name;
  OPEN new1;
  //Open the cursor, and get the value
  LoopGetRow:
  LOOP
    FETCH NEXT new1 INTO @id1, @times1, @remarks1;
    IF SQLSTATE = @err_notfound THEN
      LEAVE LoopGetRow
    END IF;
    //print the value or for other use
    PRINT (@remarks1);
  END LOOP LoopGetRow;
  CLOSE new1
END;
```

The following example replaces the myTrig trigger created in the previous example.

```
CREATE OR REPLACE TRIGGER myTrig AFTER INSERT ORDER 4 ON t0 REFERENCING NEW AS new_name FOR EACH STATEMENT
BEGIN
  FOR L1 AS new1 CURSOR FOR
    SELECT id, times, remarks FROM new_name
  DO
    //print the value or for other use
    PRINT (@remarks1);
  END FOR;
END;
```

The next example shows how you can use REFERENCING NEW in a BEFORE UPDATE trigger. This example ensures that postal codes in the new Employees table are in uppercase. You must have SELECT, ALTER, and UPDATE privileges on GROUPO.Employees to execute this statement:

```
CREATE TRIGGER emp_upper_postal_code BEFORE UPDATE OF PostalCode
```
ON GROUPO.Employees
REFERENCING NEW AS new_emp
FOR EACH ROW
WHEN ( ISNUMERIC( new_emp.PostalCode ) = 0 )
BEGIN
-- Ensure postal code is uppercase (employee might be
-- in Canada where postal codes contain letters)
SET new_emp.PostalCode = UPPER(new_emp.PostalCode)
END;

UPDATE GROUPO.Employees SET state='ON', PostalCode='n2x 4y7' WHERE
EmployeeID=191;
SELECT PostalCode FROM GROUPO.Employees WHERE EmployeeID = 191;

The next example shows how you can use REFERENCING OLD in a BEFORE DELETE trigger. This
eexample prevents deleting an employee from the Employees table who has not been terminated.

CREATE TRIGGER TR_check_delete_employee
BEFORE DELETE
ON Employees
REFERENCING OLD AS current_employee
FOR EACH ROW WHEN ( current_employee.Terminate IS NULL )
BEGIN
RAISERROR 30001 'You cannot delete an employee who has not been fired';
END;

The next example shows how you can use REFERENCING NEW and REFERENCING OLD in a
BEFORE UPDATE trigger. This example prevents a decrease in an employee's salary.

CREATE TRIGGER TR_check_salary_decrease
BEFORE UPDATE
ON GROUPO.Employees
REFERENCING OLD AS before_update
NEW AS after_update
FOR EACH ROW
BEGIN
IF after_update.salary < before_update.salary THEN
RAISERROR 30002 'You cannot decrease a salary';
END IF;
END;

The next example shows how you can use REFERENCING NEW in a BEFORE INSERT and UPDATE
trigger. The following example creates a trigger that fires before a row in the SalesOrderItems table is
inserted or updated.

CREATE TRIGGER TR_update_date
BEFORE INSERT, UPDATE
ON GROUPO.SalesOrderItems
REFERENCING NEW AS new_row
FOR EACH ROW
BEGIN
SET new_row.ShipDate = CURRENT TIMESTAMP;
END;

The following trigger displays a message on the Messages tab of the Interactive SQL Results pane
showing which action caused the trigger to fire.

CREATE TRIGGER tr BEFORE INSERT, UPDATE, DELETE
ON sample_table
REFERENCING OLD AS t1old
FOR EACH ROW
BEGIN
    DECLARE msg varchar(255);
    SET msg = 'This trigger was fired by an ';
    IF INSERTING THEN
        SET msg = msg || 'insert'
    ELSEIF DELETING THEN
        set msg = msg || 'delete'
    ELSEIF UPDATING THEN
        set msg = msg || 'update'
    END IF;
    END IF;
    MESSAGE msg TO CLIENT
END;

CREATE TRIGGER statement [T-SQL]

Creates a new trigger in the database in a manner compatible with Adaptive Server Enterprise.

Syntax 1
CREATE TRIGGER [owner.]trigger_name
ON [owner.]table_name
FOR { INSERT, UPDATE, DELETE }
AS statement-list

Syntax 2
CREATE TRIGGER [owner.]trigger_name
ON [owner.]table_name
FOR { INSERT, UPDATE }
AS
    IF UPDATE ( column-name )
        [ { AND | OR } UPDATE ( column-name ) ] ... ]
        statement-list
    [ IF UPDATE ( column-name )
        [ { AND | OR } UPDATE ( column-name ) ] ... ]
    statement-list

Remarks
CREATE TRIGGER acquires an exclusive table lock on the table.

The rows deleted or inserted are held in two temporary tables. In the Transact-SQL form of triggers, they can be accessed using the table names "deleted", and "inserted", as in Adaptive Server Enterprise. In the Watcom SQL CREATE TRIGGER statement, these rows are referenced using the REFERENCING clause.

Trigger names must be unique in the database.

Transact-SQL triggers are executed AFTER the triggering statement has executed.

Since the ORDER clause is not supported when creating Transact-SQL triggers, the value of trigger_order is set to 1. The SYSTRIGGER system table has a unique index on: table_id, event, trigger_time, and trigger_order. For a particular event (insert, update, delete), statement-level triggers are always AFTER and trigger_order cannot be set, so there can be only one of each type per table, assuming any other triggers do not set an order other than 1.
Privileges
You must be the owner of the table, or have ALTER privilege on the table, or have ALTER ANY TABLE system privilege. Additionally, you must have the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege.

Side effects
Automatic commit.

See also
- “CREATE TRIGGER statement” on page 713
- “Transact-SQL triggers” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008 Vendor extension.
- Transact-SQL ROW triggers are not supported by Adaptive Server Enterprise. The SQL Anywhere Transact-SQL dialect does not support Transact-SQL INSTEAD OF triggers, though these are supported by Adaptive Server Enterprise.

CREATE USER statement
Creates a database user or group.

Syntax
```
CREATE USER user-name [ IDENTIFIED BY password ]
[ LOGIN POLICY policy-name ]
[ FORCE PASSWORD CHANGE { ON | OFF } ]
```

Parameters
- **user-name** The name of the user you are creating.
- **IDENTIFIED BY clause** The password of the user. A user without a password cannot connect to the database.
- **policy-name** The name of the login policy to assign the user. If no login policy is specified, the DEFAULT login policy is applied.
- **FORCE PASSWORD CHANGE clause** Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in the user's policy.

Remarks
You do not have to specify a password for the user. A user without a password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID. A user ID must be a valid identifier.
User IDs cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons

Passwords are case-sensitive and they cannot:

- begin with white space, single quotes, or double quotes
- end with white space
- contain semicolons
- be longer than 255 bytes in length

Privileges

You must have the MANAGE ANY USER system privilege.

Side effects

None.

See also

- “CREATE LOGIN POLICY statement” on page 607
- “ALTER LOGIN POLICY statement” on page 445
- “ALTER USER statement” on page 508
- “COMMENT statement” on page 538
- “DROP LOGIN POLICY statement” on page 761
- “DROP USER statement” on page 786
- “Login policies” [SQL Anywhere Server - Database Administration]
- “Creating a user (Sybase Central)” [SQL Anywhere Server - Database Administration]
- “GRANT statement” on page 827
- “min_password_length option” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example creates a user named SQLTester with the password welcome. The SQLTester user will be required to specify a password when they log in.

```sql
CREATE USER SQLTester IDENTIFIED BY welcome
FORCE PASSWORD CHANGE ON;
```

CREATE VARIABLE statement

Creates a SQL variable.

Syntax

```
CREATE [ OR REPLACE ] VARIABLE identifier data-type [ { = | DEFAULT } initial-value ]
```
initial-value :
special-value
  string
  [ - ] number
  ( constant-expression )
  built-in-function ( constant-expression )
  NULL

special-value :
  CURRENT {
    DATABASE {
      DATE
      PUBLISHER
      TIME
      TIMESTAMP
      USER
      UTC TIMESTAMP
    }
    USER
  }

Parameters

**OR REPLACE clause** Specifying the OR REPLACE clause drops the named variable if it already exists and replaces its definition. You can use the OR REPLACE clause as an alternative to the VAREXISTS function in SQL scripts.

**DEFAULT clause** If you specify `initial-value`, the data type must match the type defined by `data-type`.

Remarks

The CREATE VARIABLE statement creates a new variable of the specified data type. If you specify `initial-value`, the variable is set to that value. If you do not specify an `initial-value`, the variable contains the NULL value until a different value is assigned by the SET statement.

A variable can be used in a SQL expression anywhere a column name is allowed. Name resolution is performed as follows:

1. Match any aliases specified in the query's SELECT list.
2. Match column names for any referenced tables.
3. Assume the name is a variable.

Variables belong to the current connection, and persist until you disconnect from the database or when you use the DROP VARIABLE statement. Variables are not visible to other connections. Variables are not affected by COMMIT or ROLLBACK statements.

Variables are useful for creating large text or binary objects for INSERT or UPDATE statements from embedded SQL programs.

Local variables in procedures and triggers are declared within a compound statement.

**Privileges**

None.
Side effects

None.

See also

- “Compound statements” [SQL Anywhere Server - SQL Usage]
- “BEGIN statement” on page 523
- “SQL data types” on page 89
- “DROP VARIABLE statement” on page 787
- “SET statement” on page 985
- “VAREXISTS function [Miscellaneous]” on page 406
- “Special values” on page 65

Standards and compatibility

- SQL/2008 Vendor extension.

Example

This example creates a variable named first_name, of data type VARCHAR(50).

    CREATE VARIABLE first_name VARCHAR(50);

This example creates a variable named 'birthday', of data type DATE.

    CREATE VARIABLE birthday DATE;

This example creates a variable named v1 as an INT with the initial setting of 5.

    CREATE VARIABLE v1 INT = 5;

This example creates a variable named v1 and sets its value to 10, regardless of whether or not the v1 variable already exists.

    CREATE OR REPLACE VARIABLE v1 INT = 10;

CREATE VIEW statement

Creates a view on the database.

Syntax

    CREATE [ OR REPLACE ] VIEW
    [ owner.]view-name [ ( column-name, ... ) ]
    AS query-expression
    [ WITH CHECK OPTION ]

Parameters

- OR REPLACE clause Specifying OR REPLACE (CREATE OR REPLACE VIEW) creates a view or replaces an existing view with the same name. Existing privileges are preserved when you use the OR REPLACE clause, but INSTEAD OF triggers on the view are dropped.
If you execute a CREATE OR REPLACE VIEW statement on a view that has one or more INSTEAD OF triggers, an error is returned. Drop the trigger before altering or dropping the view.

**AS clause** The SELECT statement on which the view is based. The SELECT statement must not refer to local temporary tables. Also, *query-expression* can have a GROUP BY, HAVING, WINDOW, or ORDER BY clause, and can contain UNION, EXCEPT, INTERSECT, or a common table expression.

Query semantics dictate that the order of the rows returned is undefined unless the query combines an ORDER BY clause with a TOP or FIRST clause in the SELECT statement.

**WITH CHECK OPTION clause** The WITH CHECK OPTION clause rejects any updates and inserts to the view that do not meet the criteria of the view as defined by its *query-expression*.

**Remarks**

Views do not physically exist in the database as tables. They are derived each time they are used. A view is derived as the result of a SELECT statement specified in a CREATE VIEW statement. In a view, specifying the user ID of the table owner is recommended to distinguish tables with the same name.

A view name can be used in place of a table name in SELECT, DELETE, UPDATE, and INSERT statements.

SELECT * can be used in the main query, a subquery, a derived table, or a subselect of the CREATE VIEW statement.

A query can specify a TOP n, FIRST, or LIMIT clause even if there is no ORDER BY clause. At most the specified number of rows are returned, but the order of the rows returned is not defined, so an ORDER BY could be specified but it is not required. When a view is used in a query, the ORDER BY in the view does not determine the order of rows in the query, even if there are no other tables in the FROM clause. That means that an ORDER BY should only be included in a view if it is needed to select which rows are included by a TOP n, FIRST, or LIMIT clause. Otherwise, the ORDER BY clause has no effect and it is ignored by the database server.

Views can be updated unless the *query-expression* defining the view contains a GROUP BY clause, a WINDOW clause, an aggregate function, or involves a set operator (UNION, INTERSECT, EXCEPT). An update to the view updates the underlying table(s).

The view's columns are given the names specified in the *column-name* list. If the column name list is not specified, view columns are given names from the SELECT list items. All items in the SELECT list must have unique names. To use names from the SELECT list items, each item must be a simple column name or have a specified alias.

SQL Anywhere does permit unnamed expressions in the SELECT list of the *query-expression* referenced in the CREATE VIEW statement. Unnamed expressions in the SELECT list of the *query-expression* are assigned the name *expression*, concatenated with an integer value if more than one such expression exists. For example, the following statement would define view V with three columns (expression, expression1, and expression2), and these names would appear in the SYSCOLUMN system view for the created view V.

```sql
CREATE VIEW V AS
SELECT DATEADD(DAY, 1, NOW()), DATEADD(DAY, 2, NOW()), DATEADD(DAY,
```
2, NOW() )
FROM SYS.DUMMY;

Relying on these generated names is not recommended since other views with unnamed SELECT list expressions have the identical assigned names.

Typically, a view references tables and views (and their respective attributes) that are defined in the catalog. However, a view can also reference SQL variables. In this case, when a query that references the view is executed, the value of the SQL variable is used. Views that reference SQL variables are called **parameterized views** since the variables act as parameters to the execution of the view.

Parameterized views offer an alternative to embedding the body of an equivalent SELECT block in a query as a derived table in the query's FROM clause. Parameterized views can be useful for queries that are embedded in stored procedures where the SQL variables referenced in the view are input parameters for the procedure.

It is not necessary for the SQL variable to exist when the CREATE VIEW statement is executed. However, if the SQL variable is not defined when a query that refers to the view is executed, an error is returned indicating that the column (variable) could not be found.

**Privileges**

You must have the CREATE VIEW system privilege to create views owned by you. You must have the CREATE ANY VIEW or CREATE ANY OBJECT system privilege to create views owned by others.

If the tables referenced by the view are owned by other users, you must have SELECT privileges on those tables.

**Side effects**

Automatic commit.

**See also**

- “Creating a regular view” [SQL Anywhere Server - SQL Usage]
- “ALTER VIEW statement” on page 511
- “INSTEAD OF triggers” [SQL Anywhere Server - SQL Usage]
- “SELECT statement” on page 955

**Standards and compatibility**

- **SQL/2008** CREATE VIEW is a core feature of the SQL/2008 standard, but some features of a view's embedded SELECT statement are optional language features. The ability to specify an ORDER BY clause with the top-level SELECT statement in the view definition is optional SQL/2008 language feature F852. Restricting the result set of a view using SELECT TOP or LIMIT is optional SQL/2008 language feature F859 (the SQL/2008 standard uses the FETCH clause for this purpose). Specifying WITH CHECK OPTION on a view that is not updatable—for example, the view's SELECT statement contains a derived table involving aggregation or DISTINCT, or a set operator (INTERSECT, EXCEPT, or UNION)—is optional SQL/2008 language feature T111.

Some features of CREATE VIEW are vendor extensions. Parameterized views are a vendor extension, as is the optional OR REPLACE syntax and the automatic generation of names for unnamed SELECT list expressions.
The following example creates a view showing information for male employees only. This view has the same column names as the base table:

```
CREATE VIEW MaleEmployees
AS SELECT *
FROM GROUPO.Employees
WHERE Sex = 'M';
```

The following example creates a view showing employees and the departments they belong to:

```
CREATE VIEW EmployeesAndDepartments
AS SELECT Surname, GivenName, DepartmentName
FROM GROUPO.Employees JOIN GROUPO.Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

The following example replaces the EmployeesAndDepartments view created in the previous example. After replacing the view, the view shows the city, state, and country location for each employee:

```
CREATE OR REPLACE VIEW EmployeesAndDepartments
AS SELECT Surname, GivenName, City, State, Country
FROM GROUPO.Employees JOIN GROUPO.Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

The following example creates a parameterized view based on the variables var1 and var2, which are not attributes of the Employees or Departments tables:

```
CREATE VIEW EmployeesByState
AS SELECT Surname, GivenName, DepartmentName
FROM GROUPO.Employees JOIN GROUPO.Departments
ON Employees.DepartmentID = Departments.DepartmentID
WHERE Employees.State = var1 and Employees.Status = var2;
```

Variables can appear in the view's SELECT statement in any context where a variable is a permitted expression. For example, the following parameterized view utilizes the parameter var1 as the pattern for a LIKE predicate:

```
CREATE VIEW ProductsByDescription
AS SELECT *
FROM GROUPO.Products
WHERE Products.Description LIKE var1;
```

To use this view, define the variable var1 before executing the query that references the view. For example, the following BEGIN statement could be placed in a procedure, function, or a batch statement:

```
BEGIN
DECLARE var1 CHAR(20);
SET var1 = '%cap%';
SELECT * FROM ProductsByDescription
END
```

**DEALLOCATE DESCRIPTOR statement [ESQL]**

Frees memory associated with a SQL descriptor area.
**DEALLOCATE DESCRIPTOR statement**

This statement has no effect in SQL Anywhere, and is ignored. It is provided for compatibility with Adaptive Server Enterprise and Microsoft SQL Server. Refer to your Adaptive Server Enterprise or Microsoft SQL Server documentation for more information about this statement.

**Standards and compatibility**
- **SQL/2008**  Vendor extension.

---

**Declaration section [ESQL]**

Declares host variables in an embedded SQL program. Host variables are used to exchange data with the database.
Syntax

```c
EXEC SQL BEGIN DECLARE SECTION;
C declarations
EXEC SQL END DECLARE SECTION;
```

Remarks

A declaration section is simply a section of C variable declarations surrounded by the BEGIN DECLARE
SECTION and END DECLARE SECTION statements. A declaration section makes the SQL
preprocessor aware of C variables that are used as host variables. Not all C declarations are valid inside a
declaration section.

Privileges

None.

See also

- “Host variables in embedded SQL” [SQL Anywhere Server - Programming]
- “BEGIN statement” on page 523

Standards and compatibility

- SQL/2008 Core feature.

Example

```c
EXEC SQL BEGIN DECLARE SECTION;
char *surname, initials[5];
int dept;
EXEC SQL END DECLARE SECTION;
```

DECLARE CURSOR statement [ESQL] [SP]

Declares a cursor.

Syntax 1 [ESQL]

```sql
DECLARE cursor-name
[ UNIQUE ]
[ NO SCROLL
  | DYNAMIC SCROLL
  | SCROLL
  | INSENSITIVE
  | SENSITIVE
]
CURSOR FOR
{ select-statement
  | statement-name
  | call-statement
}
```

Syntax 2 [SP]

```sql
DECLARE cursor-name
[ NO SCROLL
  | DYNAMIC SCROLL
```
### SQL statements

```sql
CURSOR
{ FOR select-statement
  FOR call-statement
  USING variable-name }

cursor-name : identifier

statement-name : identifier | hostvar

variable-name : identifier

Parameters

**UNIQUE clause**  When a cursor is declared UNIQUE, the query is forced to return all the columns required to uniquely identify each row. Often this means ensuring that all columns in the primary key or a uniqueness table constraint are returned. Any columns that are required but were not specified in the query are added to the result set.

A DESCRIBE done on a UNIQUE cursor sets the following additional options in the indicator variables:

- **DT_KEY_COLUMN**  The column is part of the key for the row.
- **DT_HIDDEN_COLUMN**  The column was added to the query because it was required to uniquely identify the rows.

**NO SCROLL clause**  A cursor declared NO SCROLL is restricted to moving forward through the result set using FETCH NEXT and FETCH RELATIVE 0 seek operations.

As rows cannot be returned to once the cursor leaves the row, there are no sensitivity restrictions on the cursor. When a NO SCROLL cursor is requested, SQL Anywhere supplies the most efficient kind of cursor, which is an insensitive cursor.

**DYNAMIC SCROLL clause**  DYNAMIC SCROLL is the default cursor type. DYNAMIC SCROLL cursors can use all formats of the FETCH statement.

When a DYNAMIC SCROLL cursor is requested, SQL Anywhere supplies an insensitive cursor. When using cursors there is always a trade-off between efficiency and consistency. Insensitive cursors provide efficient performance at the expense of consistency.

**SCROLL clause**  A cursor declared SCROLL can use all formats of the FETCH statement. When a SCROLL cursor is requested, SQL Anywhere supplies a value-sensitive cursor. With a value-sensitive cursor, a subsequent fetch of a previously fetched result row may return a warning or an error if the underlying row has been modified or deleted.

SQL Anywhere must execute value-sensitive cursors in such a way that result set membership is guaranteed. DYNAMIC SCROLL cursors are more efficient and should be used unless the consistent behavior of SCROLL cursors is required.

**INSENSITIVE clause**  A cursor declared INSENSITIVE has its membership fixed when it is opened; a temporary table is created with a copy of all the original rows. FETCHING from an INSENSITIVE cursor
does not see the effect of any other INSERT, UPDATE, or DELETE statement from concurrently-executing transactions, or any other update operations from within the same transaction. INSENSITIVE cursors are not updatable.

**SENSITIVE clause**  A cursor declared SENSITIVE is sensitive to changes to membership or values of the result set.

**FOR statement-name clause**  Statements are named using the PREPARE statement. Cursors can be declared only for a prepared SELECT or CALL. The cursor updatability specified in the PREPARE statement is used for the cursor, unless the SQL preprocessor -m HISTORICAL option is specified.

**USING variable-name clause**  For use within stored procedures only. The variable is a string containing a SELECT statement for the cursor. The variable must be available when the DECLARE is processed, and so must be either a parameter to the procedure, or nested inside another BEGIN...END after the variable has been assigned a value.

**Remarks**

Cursors are the primary means for manipulating the results of queries. The DECLARE CURSOR statement declares a cursor with the specified name for a SELECT statement or a CALL statement. In a Watcom SQL procedure, trigger, or batch, a DECLARE CURSOR statement must appear with other declarations, immediately following the BEGIN keyword. Cursor names must be unique.

If a cursor is declared inside a compound statement, it exists only for the duration of that compound statement (whether it is declared in a Watcom SQL or Transact-SQL compound statement).

When a single statement is processed, all of the DECLARE CURSOR statements must use distinct names, even if the cursors are declared in scopes that do not overlap. However, the cursor can only be used within the compound statement that declares it.

The type of cursor specified in a DECLARE CURSOR statement can dictate the execution plan selected by the query optimizer for that statement. For example, an INSENSITIVE cursor over a SELECT statement requires the complete materialization of the result set of the SELECT statement when the cursor is opened. Moreover, the type of cursor must match the characteristics of the underlying statement. If there is a mismatch between the cursor type and the statement, then the cursor type may be changed automatically. For example, an INSENSITIVE cursor declaration conflicts with an updatable SELECT statement that specifies FOR UPDATE, since by definition INSENSITIVE cursors are read only. In this case, the cursor type is changed automatically from INSENSITIVE to an updatable, value-sensitive cursor when the cursor is opened.

If the updatability of a SELECT statement embedded in a cursor declaration is unspecified, it is determined by the setting of the ansi_update_constraints option.

**Privileges**

None.

**Side effects**

If the cursor type must be changed to satisfy the requirements of the underlying statement, a warning is returned when the cursor is opened.
Standards and compatibility

- **SQL/2008**  DECLARE CURSOR is a core feature of the SQL/2008 standard. The ability to specify FOR UPDATE with SCROLL or NO SCROLL is optional SQL language feature F831, "Full cursor update". Using DECLARE CURSOR in an embedded SQL program constitutes optional SQL language feature B031. Some cursor types are also optional SQL features. These include:
  - INSENSITIVE cursors are optional SQL language feature F791 of the SQL/2008 standard.
  - SENSITIVE cursors are optional SQL language feature F231 of the SQL/2008 standard.
  - Scrollable cursors are optional SQL language feature F431 of the SQL/2008 standard.

SQL Anywhere supports a number of vendor extensions to DECLARE CURSOR, including:
  - SQL Anywhere supports several extensions to the FOR UPDATE clause, which SQL/2008 defines as a clause of the DECLARE CURSOR statement.
  - WITH HOLD is specified as a clause of the OPEN statement, rather than as a clause of the DECLARE CURSOR statement as defined in SQL/2008.
  - The SQL/2008 standard separates the notions of cursor sensitivity and scrollability, while for historical reasons SQL Anywhere combines the two. In SQL Anywhere, all cursors are forward and backward scrollable except for those declared as NO SCROLL.
  - DYNAMIC SCROLL and UNIQUE are vendor extensions. DYNAMIC SCROLL has similar behavior to cursors declared as ASENSITIVE in the SQL/2008 standard.
  - The ability to declare a cursor over a CALL statement, or with a USING clause, is a vendor extension.

- **Transact-SQL**  DECLARE CURSOR is supported by Adaptive Server Enterprise, but there are several behavioral differences. Adaptive Server Enterprise differentiates, as in SQL/2008, between scrollability and sensitivity; in Adaptive Server Enterprise, cursor sensitivity options are SEMI-
SENSITIVE, INSENSITIVE, or default (akin to ASENSITIVE). In Adaptive Server Enterprise, NO SCROLL cursors are the default, and all scrollable cursors are read-only. Several features of the DECLARE CURSOR statement are not supported by Adaptive Server Enterprise. These include:

- Adaptive Server Enterprise does not support the SQL Anywhere cursor concurrency clause. To acquire a lock on a fetched row, you must use the HOLDLOCK table hint.

- Adaptive Server Enterprise does not support DYNAMIC SCROLL or UNIQUE cursors. DYNAMIC SCROLL is similar to Adaptive Server Enterprise default cursor behavior.

- The ability to declare a cursor over a CALL statement, or with a USING clause, is not supported by Adaptive Server Enterprise.

In Adaptive Server Enterprise, Transact-SQL procedures and functions can contain multiple DECLARE CURSOR statements that use the same cursor name. In Adaptive Server Enterprise, the DEALLOCATE CURSOR statement is used to eliminate a cursor from the current scope, so that a subsequent OPEN statement can reference the correct, previously-declared cursor. This feature is not supported in SQL Anywhere. In SQL Anywhere, all cursors in a given scope must have unique names. If a Transact-SQL dialect procedure contains multiple cursor declarations with the same name, the procedure parses without error. However, at execution time, if a second DECLARE CURSOR statement with the same cursor name is executed, an error occurs.

You should be aware that the TDS wire protocol for Open Client and jConnect connections does not implement true scrollable result sets. When scrolling backward through a cursor, the FETCH request may be satisfied immediately if the desired row is within a window of prefetched rows that have already been retrieved by the TDS client. If the desired row is beyond this window, however, the cursor's SELECT statement may be re-executed.

Example

The following example illustrates how to declare a scroll cursor in embedded SQL:

```sql
EXEC SQL DECLARE cur_employee SCROLL CURSOR
FOR SELECT * FROM GROUPO.Employees;
```

The following example illustrates how to declare a cursor for a prepared statement in embedded SQL:

```sql
EXEC SQL PREPARE employee_statement
FROM 'SELECT Surname FROM GROUPO.Employees'FOR READ ONLY;
EXEC SQL DECLARE cur_employee CURSOR
FOR employee_statement;
```

The following example illustrates the use of cursors in a stored procedure:

```sql
BEGIN
  DECLARE cur_employee CURSOR FOR
  SELECT Surname
  FROM GROUPO.Employees;
  DECLARE name CHAR(40);
  OPEN cur_employee;
  lp: LOOP
  FETCH NEXT cur_employee INTO name;
  IF SQLCODE <> 0 THEN LEAVE lp END IF;
  ...
```
This example shows the USING clause being used as a parameter to the procedure:

```
CREATE FUNCTION GetRowCount( IN qry LONG VARCHAR )
RETURNS INT
BEGIN
  DECLARE crsr CURSOR USING qry;
  DECLARE rowcnt INT;
  SET rowcnt = 0;
  OPEN crsr;
  lp: LOOP
    FETCH crsr;
    IF SQLCODE <> 0 THEN LEAVE lp END IF;
    SET rowcnt = rowcnt + 1;
  END LOOP;
  CLOSE crsr;
  RETURN rowcnt;
END;
```

This example shows the USING clause nested inside a BEGIN...END, after `variable-name` has been assigned a value.

```
CREATE PROCEDURE get_table_name(
  IN id_value INT, OUT tabname CHAR(128)
) BEGIN
  DECLARE qry LONG VARCHAR;
  SET qry = 'SELECT table_name FROM SYS.SYSTAB ' ||
    'WHERE table_id=' || string(id_value);
  BEGIN
    DECLARE crsr CURSOR USING qry;
    OPEN crsr;
    FETCH crsr INTO tabname;
    CLOSE crsr;
    RETURN;
  END
END;
```

The following example returns an error as the two cursor names within the statement are not unique:

```
BEGIN
  BEGIN
    DECLARE MyCursor DYNAMIC SCROLL CURSOR FOR SELECT 1;
  END;
  BEGIN
    DECLARE MyCursor DYNAMIC SCROLL CURSOR FOR SELECT 2;
  END;
END;
```

The following example returns an error since the cursor has not been declared within the statement that is trying to open it.

```
BEGIN
  BEGIN
    DECLARE MyCursor DYNAMIC SCROLL CURSOR FOR SELECT 1;
  END;
  BEGIN
    OPEN MyCursor;
  END;
END;
```
DECLARE LOCAL TEMPORARY TABLE statement

Declares a local temporary table.

Syntax

```
DECLARE LOCAL TEMPORARY TABLE table-name
( { column-definition [ column-constraint ... ] | table-constraint | pctfree }, ... )
[ ON COMMIT { DELETE | PRESERVE } ROWS
 | NOT TRANSACTIONAL ]

pctfree : PCTFREE percent-free-space
percent-free-space : integer
```

Parameters

**ON COMMIT clause**  
By default, the rows of a temporary table are deleted on a COMMIT. You can use the ON COMMIT clause to preserve rows on a COMMIT.

**NOT TRANSACTIONAL clause**  
A table created using this clause is not affected by either COMMIT or ROLLBACK. The NOT TRANSACTIONAL clause provides performance improvements in some circumstances because operations on non-transactional temporary tables do not cause entries to be made in the rollback log. For example, NOT TRANSACTIONAL can be useful if procedures that use the temporary table are called repeatedly with no intervening COMMITs or ROLLBACKs.

Remarks

You cannot use the REFERENCES column-constraint or the FOREIGN KEY table-constraint on a local temporary table.

The DECLARE LOCAL TEMPORARY TABLE statement declares a temporary table.

Tables created using DECLARE LOCAL TEMPORARY TABLE do not appear in the SYSTABLE view of the system catalog.

The rows of a declared temporary table are deleted when the table is explicitly dropped or when the table goes out of scope. You can also explicitly delete rows using TRUNCATE or DELETE.

Declared local temporary tables within compound statements exist within the compound statement. Otherwise, the declared local temporary table exists until the end of the connection.

Two local temporary tables within the same scope cannot have the same name. If you create temporary table with the same name as a base table, the base table only becomes visible within the connection once the scope of the local temporary table ends. A connection cannot create a base table with the same name as an existing temporary table.

If you want a procedure to create a local temporary table that persists after the procedure completes, use the CREATE LOCAL TEMPORARY TABLE statement instead.
Privileges

None.

Side effects

None.

See also

- “CREATE TABLE statement” on page 690
- “CREATE LOCAL TEMPORARY TABLE statement” on page 606
- “Compound statements” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- **SQL/2008** DECLARE LOCAL TEMPORARY TABLE is part of optional language feature F531 of the SQL/2008 standard. The PCTFREE and NOT TRANSACTIONAL clauses are vendor extensions. The column and constraint definitions defined by the statement may also include vendor extension syntax. In SQL/2008, the standard stipulates that tables created via the DECLARE LOCAL TEMPORARY TABLE statement appear in the system catalog; this is not the case with SQL Anywhere.

- **Transact-SQL** DECLARE LOCAL TEMPORARY TABLE is not supported by Adaptive Server Enterprise. In Sybase Adaptive Server Enterprise, one creates a temporary table using the CREATE TABLE statement with a table name that begins with the special character #.

Example

The following example illustrates how to declare a temporary table in a stored procedure:

```
BEGIN
  DECLARE LOCAL TEMPORARY TABLE TempTab ( number INT );
  ... 
END
```

**DECLARE statement**

Declares a SQL variable or an exception within a compound statement (BEGIN...END).

Syntax 1 - Declaring a variable

```
DECLARE variable-name [, ... ] data-type
[ { = | DEFAULT } initial-value ]
```

- initial-value:
- special-value
  - string
  - number
  - ( constant-expression )
  - built-in-function ( constant-expression )
  - NULL
Syntax 2 - Declaring an exception

```
DECLARE exception-name EXCEPTION
FOR SQLSTATE [ VALUE ] string
```

Remarks

**DECLARE variable-name**: Variables used in the body of a procedure, trigger, or batch can be declared using the DECLARE statement using Syntax 1. The variable persists for the duration of the compound statement in which it is declared. If you specify `initial-value`, the variable is set to that value. If you do not specify an `initial-value`, the variable contains the NULL value until a different value is assigned by the SET statement.

The body of a Watcom SQL procedure or trigger is a compound statement, and variables must be declared with other declarations, such as a cursor declaration (DECLARE CURSOR), immediately following the BEGIN keyword. In a Transact-SQL procedure or trigger, there is no such restriction.

If you specify `initial-value`, the data type must match the type defined by `data-type`.

**DECLARE exception-name EXCEPTION**: Use this syntax to declare variables for SQL language exceptions within a compound statement (BEGIN...END). The variables can be used, for example, for comparison with the SQLSTATEs obtained during execution, with the SIGNAL statement, or as part of the exception case within the exception handler.

Privileges

None.

Side effects

None.

See also

- “SQL data types” on page 89
- “DECLARE statement” on page 736
- “DECLARE CURSOR statement [ESQL] [SP]” on page 729
- “BEGIN statement” on page 523
- “SQLSTATE special value” on page 74
- “Special values” on page 65
- “Exception handlers” [SQL Anywhere Server - SQL Usage]
Standards and compatibility

- **SQL/2008 Syntax 1** (declaring variables) - Persistent Stored Module feature. **Syntax 2** (declaring exceptions) - The form of exception declaration supported by SQL Anywhere, namely the DECLARE EXCEPTION statement, is a vendor extension; in SQL/2008, exceptions are specified using a handler declaration using the keywords DECLARE HANDLER. The DECLARE...EXCEPTION syntax is not allowed in T-SQL procedures.

- **Transact-SQL Syntax 2** (declaring exceptions) cannot be used in Transact-SQL compound statements and procedures.

Example

The following batch illustrates the use of the DECLARE statement and prints a message in the database server messages window:

```
BEGIN
  DECLARE varname CHAR(61);
  SET varname = 'Test name';
  MESSAGE varname;
END
```

This example declares the following variables:

- v1 as an INT with the initial setting of 5.
- v2 and v3 as CHAR(10), both with an initial value of abc.

```
BEGIN
  DECLARE v1 INT = 5;
  DECLARE v2, v3 CHAR(10) = 'abc';
  // ...
END
```

The following procedure declares an exception for use with the SQLSTATE comparison:

```
CREATE PROCEDURE HighSales (IN cutoff INT, OUT HighValues INT)
BEGIN
  DECLARE err_notfound EXCEPTION FOR SQLSTATE '02000';
  DECLARE curThisCust CURSOR FOR
    SELECT CAST( sum( SalesOrderItems.Quantity * Products.UnitPrice ) AS INTEGER) VALUE
    FROM Customers
    LEFT OUTER JOIN SalesOrders
    LEFT OUTER JOIN SalesOrderItems
    LEFT OUTER JOIN Products
    GROUP BY CompanyName;
  DECLARE ThisValue INT;
  SET HighValues = 0;
  OPEN curThisCust;
  CustomerLoop:
  LOOP
    FETCH NEXT curThisCust
    INTO ThisValue;
    IF SQLSTATE = err_notfound THEN
      LEAVE CustomerLoop;
    END IF;
    IF ThisValue > cutoff THEN
      SET HighValues = HighValues + ThisValue;
    END IF;
```
The following compound statement declares an exception for use with SIGNAL and an exception handler:

```
BEGIN
  DECLARE err_div_by_0 EXCEPTION FOR
    SQLSTATE '22012';
  DECLARE curQuantity CURSOR FOR
    SELECT Quantity
    FROM SalesOrderItems
    WHERE ProductID = 300;
  DECLARE Quantities INT;
  DECLARE altogether INT;
  SET Quantities = 0;
  SET altogether = 0;
  OPEN curQuantity;
  LOOP
    FETCH NEXT curQuantity
    INTO Quantities;
    IF SQLSTATE = '02000' THEN
      SIGNAL err_div_by_0;
    END IF;
    SET altogether = altogether + Quantities;
  END LOOP;
  EXCEPTION
    WHEN err_div_by_0 THEN
      CLOSE curQuantity;
      SELECT altogether;
      RETURN;
    WHEN OTHERS THEN
      RESIGNAL;
  END;
```

**DELETE statement (positioned) [ESQL] [SP]**

Deletes the data at the current location of a cursor.

**Syntax**

```
DELETE [ [FROM ]table ] WHERE CURRENT OF cursor-name
```

- `cursor-name`: identifier | hostvar
- `table`: [ owner.]table-or-view [ [ AS ] correlation-name ]
- `owner`: identifier
- `table-or-view`: identifier
- `correlation-name`: identifier

**Remarks**

This form of the DELETE statement deletes the current row of the specified cursor. The current row is defined to be the last row fetched from the cursor.
The table from which rows are deleted is determined as follows:

- If no FROM clause is included, the cursor must be on a single table only.

- If the cursor is for a joined query (including using a view containing a join), then the FROM clause must be used. Only the current row of the specified table is deleted. The other tables involved in the join are not affected.

- If a FROM clause is included, table must unambiguously identify an updatable table in the cursor. If acorrelation-name is specified, the server attempts to match that correlation name with a correlation name specified in the underlying cursor. If a correlation name is not specified in the DELETE statement, and a table owner is not specified, then the server attempts to match table-or-view with an updatable table in the underlying cursor. table-or-view is first matched against any correlation names.
  
  □ If a correlation name exists in the underlying cursor, table-or-view may be matched with the corresponding correlation name.
  
  □ If a correlation name does not exist, table-or-view must unambiguously match a table name in the cursor.

- If a FROM clause is included, and a table owner is specified, table must unambiguously match an updatable table in the underlying cursor.

- The positioned DELETE statement can be used on a cursor open on a view as long as the view is updatable.

**Privileges**

You must be the table owner, or have DELETE privilege on the table.

**Side effects**

None.

**See also**

- “UPDATE statement” on page 1037
- “UPDATE (positioned) statement [ESQL] [SP]” on page 1032
- “INSERT statement” on page 860
- “PUT statement [ESQL]” on page 919

**Standards and compatibility**

- **SQL/2008** The DELETE statement (positioned) is a core feature of the SQL/2008 standard. The ability to use a positioned DELETE statement from within an embedded SQL program is part of optional SQL language feature B031, "Basic dynamic SQL".

  The FROM keyword is mandatory in SQL/2008, but optional in SQL Anywhere. The range of cursors that can be updated may contain vendor extensions if the ansi_update_constraints option is set to Off.

**Example**

The following statement removes the current row in the cursor cur_employee from the database.
DELETE
WHERE CURRENT OF cur_employee;

DELETE statement

Deletes rows from the database.

Syntax 1

DELETE [ row-limitation ]
[ FROM ] [ owner. ] table-or-view [ [ AS ] correlation-name ]
[ WHERE search-condition ]
[ ORDER BY { expression | integer } [ ASC | DESC ], ... ]
[ OPTION( query-hint, ... ) ]

Syntax 2 - Transact-SQL

DELETE [ row-limitation ]
[ FROM ] [ owner. ] table-or-view [ [ AS ] correlation-name ]
[ FROM table-expression ]
[ WHERE search-condition ]
[ ORDER BY { expression | integer } [ ASC | DESC ], ... ]
[ OPTION( query-hint, ... ) ]

table-or-view : identifier

row-limitation :
FIRST
| TOP { ALL | limit-expression } [ START AT startat-expression ]

limit-expression : simple-expression
startat-expression : simple-expression

simple-expression :
integer
| variable
| ( simple-expression )
| ( simple-expression { + | - | * } simple-expression )

query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| FORCE NO OPTIMIZATION
| option-name = option-value

table-expression : A full table expression that can include joins. See “FROM clause” on page 810.

option-name : identifier

option-value :
hostvar (indicator allowed)
| string
| identifier
| number
Parameters

**row-limitation clause**  The row limitation clause allows you to restrict the rows being deleted to only a subset of the rows that satisfy the WHERE clause. The TOP and START AT arguments can be simple arithmetic expressions over host variables, integer constants, or integer variables. The TOP argument must evaluate to a value greater than or equal to 0. The START AT argument must evaluate to a value greater than 0. When specifying these clauses, an ORDER BY clause is required to order the rows in a meaningful manner.

**FROM clause**  The FROM clause indicates the table from which rows will be deleted. In Syntax 2, the second FROM clause in the DELETE statement determines the rows to be deleted from the specified table based on joins with other tables. *table-expression* can contain arbitrarily complex table expressions, including derived tables and KEY and NATURAL joins.

The following examples illustrate how correlation names are matched when Syntax 2 is used. With this statement:

```sql
DELETE
FROM table_1
FROM table_1 AS alias_1, table_2 AS alias_2
WHERE ...
```

table table_1 doesn't have a correlation name in the first FROM clause but does in the second FROM clause. In this case, table_1 in the first clause is identified with alias_1 in the second clause — there is only one instance of table_1 in this statement. This is allowed as an exception to the general rule that where a table is identified with a correlation name and without a correlation name in the same statement, two instances of the table are considered.

However, in the following example, there are two instances of table_1 in the second FROM clause. The statement fails with a syntax error because it is not clear which instance of the table_1 from the second FROM clause matches the first instance of table_1 in the first FROM clause.

```sql
DELETE
FROM table_1
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

**WHERE clause**  The DELETE statement deletes all the rows that satisfy the conditions in the WHERE clause. If no WHERE clause is specified, all rows from the named table are deleted. If a second FROM clause is present, the WHERE clause qualifies the rows of the second FROM clause's *table-expression*.

**ORDER BY clause**  Specifies the sort order for the rows to be deleted. Normally, the order in which rows are updated does not matter. However, with the FIRST or TOP clause, the order can be significant.

You cannot use ordinal column numbers in the ORDER BY clause.

Each item in the ORDER BY list can be labeled as ASC for ascending order (the default) or DESC for descending order.
OPTION clause  Use this clause to specify hints for executing the statement. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION option-value
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- option-name = option-value. An OPTION( isolation_level = ... ) specification in the query text overrides all other means of specifying isolation level for a query.

Remarks
Deleting a significant amount of data using the DELETE statement causes an update to column statistics.

To delete all of the rows of a table, consider using the more efficient TRUNCATE TABLE statement.

DELETE operations can be performed on views if the query specification defining the view is updatable. A view is updatable provided the SELECT statement defining the view has only one table in the FROM clause and does not contain a DISTINCT clause, a GROUP BY clause, a WINDOW clause, an aggregate function, or involve a set operator such as UNION or INTERSECT.

Privileges
You must be the table owner, or have SELECT and DELETE privileges on the table.

Side effects
None.

See also
- “TRUNCATE statement” on page 1019
- “Row limitation clauses in SELECT, UPDATE, and DELETE query blocks” [SQL Anywhere Server - SQL Usage]
- “INSERT statement” on page 860
- “INPUT statement [Interactive SQL]” on page 854
- “FROM clause” on page 810
- “OPTION clause, SELECT statement” on page 962
- “Regular views” [SQL Anywhere Server - SQL Usage]
- “Locks during deletes” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- **SQL/2008**  Syntax 1 is a core feature of the SQL/2008 standard, whereas Syntax 2 is a Transact-SQL vendor extension. The following features of Syntax 1 are vendor extensions:
  - The optional FROM keyword.
  - The row-limitation clause and the ORDER BY clause.
  - The OPTION clause.

Example
Remove all data before 2000 from the FinancialData table.
DELETE
FROM GROUPO.FinancialData
WHERE Year < 2000;

Remove the first 10 orders from SalesOrderItems table where ship date is older than 2001-01-01 and their region is Central.

DELETE TOP 10
FROM GROUPO.SalesOrderItems
FROM GROUPO.SalesOrders
WHERE SalesOrderItems.ID = SalesOrders.ID
and ShipDate < '2001-01-01' and Region ='Central'
ORDER BY ShipDate ASC;

Remove department 600 from the database, executing the statement at isolation level 3.

DELETE FROM GROUPO.Departments
WHERE DepartmentID = 600
OPTION( isolation_level = 3 );

**DESCRIBE statement [ESQL]**

Gets information about the host variables required to store data retrieved from the database, or host variables required to pass data to the database.

**Syntax**

```
DESCRIBE
[ USER TYPES ]
[ ALL | BIND VARIABLES FOR | INPUT | OUTPUT
SELECT LIST FOR ]
[ LONG NAMES | long-name-spec | WITH VARIABLE RESULT ]
[ FOR ] { statement-name | CURSOR cursor-name }
INTO sqlda-name
```

- `long-name-spec`: OWNER.TABLE.COLUMN | TABLE.COLUMN | COLUMN
- `statement-name`: identifier | hostvar
- `cursor-name`: declared cursor
- `sqlda-name`: identifier

**Parameters**

**USER TYPES clause** A DESCRIBE statement with the USER TYPES clause returns information about domains of a column. Typically, such a DESCRIBE is done when a previous DESCRIBE returns an indicator of DT_HAS_USERTYPE_INFO.

The information returned is the same as for a DESCRIBE without the USER TYPES keywords, except that the sqlname field holds the name of the domain, instead of the name of the column.

If the DESCRIBE uses the LONG NAMES clause, the sqldata field holds this information.
**ALL clause**  
DESCRIBE ALL allows you to describe INPUT and OUTPUT with one request to the database server. This has a performance benefit. The OUTPUT information is filled in the SQLDA first, followed by the INPUT information. The sqlda field contains the total number of INPUT and OUTPUT variables. The DT_DESCRIBE_INPUT bit in the indicator variable is set for INPUT variables and clear for OUTPUT variables.

**BIND VARIABLES FOR clause**  
Equivalent to the INPUT clause.

**SELECT LIST FOR clause**  
Equivalent to the OUTPUT clause.

**INPUT clause**  
A bind variable is a value supplied by the application when the database executes the statements. Bind variables can be considered parameters to the statement. DESCRIBE INPUT fills in the name fields in the SQLDA with the bind variable names. DESCRIBE INPUT also puts the number of bind variables in the sqlda field of the SQLDA.

DESCRIBE uses the indicator variables in the SQLDA to provide additional information. DT_PROCEDURE_IN and DT_PROCEDURE_OUT are bits that are set in the indicator variable when a CALL statement is described. DT_PROCEDURE_IN indicates an IN or INOUT parameter and DT_PROCEDURE_OUT indicates an INOUT or OUT parameter. Procedure RESULT columns will have both bits clear. After a describe OUTPUT, these bits can be used to distinguish between statements that have result sets (need to use OPEN, FETCH, RESUME, CLOSE) and statements that do not (need to use EXECUTE). DESCRIBE INPUT only sets DT_PROCEDURE_IN and DT_PROCEDURE_OUT appropriately when a bind variable is an argument to a CALL statement; bind variables within an expression that is an argument in a CALL statement will not set the bits.

**OUTPUT clause**  
The DESCRIBE OUTPUT statement fills in the data type and length for each SELECT list item in the SQLDA. The name field is also filled in with a name for the SELECT list item. If an alias is specified for a SELECT list item, the name will be that alias. Otherwise, the name is derived from the SELECT list item: if the item is a simple column name, it is used; otherwise, a substring of the expression is used. DESCRIBE will also put the number of SELECT list items in the sqlda field of the SQLDA.

If the statement being described is a UNION of two or more SELECT statements, the column names returned for DESCRIBE OUTPUT are the same column names which would be returned for the first SELECT statement.

If you describe a CALL statement, the DESCRIBE OUTPUT statement fills in the data type, length, and name in the SQLDA for each INOUT or OUT parameter in the procedure. DESCRIBE OUTPUT also puts the number of INOUT or OUT parameters in the sqlda field of the SQLDA.

If you describe a CALL statement with a result set, the DESCRIBE OUTPUT statement fills in the data type, length, and name in the SQLDA for each RESULT column in the procedure definition. DESCRIBE OUTPUT will also put the number of result columns in the sqlda field of the SQLDA.

**LONG NAMES clause**  
The LONG NAMES clause is provided to retrieve column names for a statement or cursor. Without this clause, there is a 29-character limit on the length of column names; with the clause, names of an arbitrary length are supported.

If LONG NAMES is used, the long names are placed into the SQLDATA field of the SQLDA, as if you were fetching from a cursor. None of the other fields (SQLLEN, SQLTYPE, and so on) are filled in. The
SQLDA must be set up like a FETCH SQLDA: it must contain one entry for each column, and the entry must be a string type. If there is an indicator variable, truncation is indicated in the usual fashion.

The default specification for the long names is **TABLE.COLUMN**.

**WITH VARIABLE RESULT clause** This clause is used to describe procedures that can have more than one result set, with different numbers or types of columns.

If WITH VARIABLE RESULT is used, the database server sets the SQLCOUNT value after the DESCRIBE statement to one of the following values:

- **0** The result set may change. The procedure call should be described again following each OPEN statement.
- **1** The result set is fixed. No re-describing is required.

**Remarks**

The DESCRIBE statement sets up the named SQLDA to describe either the OUTPUT (equivalently SELECT LIST FOR) or the INPUT (equivalently BIND VARIABLES FOR) for the named statement.

In the INPUT case, DESCRIBE BIND VARIABLES does not set up the data types in the SQLDA: this needs to be done by the application. The ALL keyword allows you to describe INPUT and OUTPUT in one SQLDA.

If you specify a statement name, the statement must have been previously prepared using the PREPARE statement with the same statement name and the SQLDA must have been previously allocated.

If you specify a cursor name, the cursor must have been previously declared and opened. The default action is to describe the OUTPUT. Only SELECT statements and CALL statements have OUTPUT. A DESCRIBE OUTPUT on any other statement, or on a cursor that is not a dynamic cursor, indicates no output by setting the sqld field of the SQLDA to zero.

In embedded SQL, NCHAR, NVARCHAR and LONG NVARCHAR are described as DT_FIXCHAR, DT_VARCHAR, and DT_LONGVARCHAR, respectively, by default. If the db_change_nchar_charset function has been called, these data types are described as DT_NFIXCHAR, DT_NVARCHAR and DT_LONGNVARCHAR, respectively.

**Privileges**

None.

**Side effects**

None.
Standards and compatibility

- **SQL/2008** The DESCRIBE OUTPUT statement is optional SQL language feature B031, "Basic dynamic SQL", of the SQL/2008 standard. The DESCRIBE INPUT statement is optional SQL language feature B032, "Extended dynamic SQL". Many of the other clauses of the DESCRIBE statement are vendor extensions. These include:
  - The USER TYPES, ALL, BIND VARIABLES FOR, LONG NAMES, and WITH VARIABLE RESULT clauses.
  - DESCRIBE uses the INTO clause to identify the sqlda; in the SQL/2008 standard, the USING keyword is used instead.
  - In the SQL/2008 standard, the CURSOR clause ends with the keyword STRUCTURE. STRUCTURE is not supported by SQL Anywhere.

Example

The following example shows how to use the DESCRIBE statement:

```sql
sqlda = alloc_sqlda( 3 );
EXEC SQL DESCRIBE OUTPUT
FOR employee_statement
  INTO sqlda;
if( sqlda->sqld > sqlda->sqln ) {
  actual_size = sqlda->sqld;
  free_sqlda( sqlda );
  sqlda = alloc_sqlda( actual_size );
  EXEC SQL DESCRIBE OUTPUT
    FOR employee_statement
      INTO sqlda;
}
```

DESCRIBE statement [Interactive SQL]

Returns information about a given database object.

Syntax 1 - Describing database objects

```
DESCRIBE [ [ INDEX FOR ] TABLE | PROCEDURE ] [ owner. ] object-name

object-name : table
```
Syntax 2 - Describing the current connection

DESCRIBE CONNECTION

Parameters

INDEX FOR clause  Indicates that you want to see the indexes for the specified object-name.

TABLE clause  Indicates that object-name to be described is a table or a view.

PROCEDURE clause  Indicates that object-name is a procedure or a function.

Remarks

Use DESCRIBE TABLE to list all the columns in the specified table or view. The DESCRIBE TABLE statement returns one row per table column, containing:

- Column  The name of the column.
- Type  The type of data in the column.
- Nullable  Whether nulls are allowed (1=yes, 0=no).
- Primary Key  Whether the column is in the primary key (1=yes, 0=no).

Use DESCRIBE INDEX FOR TABLE to list all the indexes for the specified table. The DESCRIBE TABLE statement returns one row per index, containing:

- Index Name  The name of the index.
- Columns  The columns in the index.
- Unique  Whether the index is unique (1=yes, 0=no).
- Type  The type of index. Possible values are: Clustered, Statistic, Hashed, and Other.

Use DESCRIBE PROCEDURE to list all the parameters used by the specified procedure or function. The DESCRIBE PROCEDURE statement returns one row for each parameter, containing:

- Parameter  The name of the parameter.
- Type  The data type of the parameter.
- In/Out  Information about what is passed to, or returned from, the parameter. Possible values are:
  - In  The parameter is passed to the procedure, but is not modified.
  - Out  The procedure ignores the parameter's initial value and sets its value when the procedure returns.
○ **In/Out**  The parameter is passed to the procedure and the procedure sets the parameter's value when the procedure returns.

○ **Result**  The parameter returns a result set.

○ **Return**  The parameter returns a declared return value.

If you do not specify either TABLE or PROCEDURE (for example, DESCRIBE `object-name`), Interactive SQL assumes the object is a table. However, if no such table exists, Interactive SQL attempts to describe the object as either a procedure or a function.

Use Syntax 2 to list information about the database or database server that Interactive SQL is connected to. The following properties are returned:

- **Database Product**  The name and version number of the database product Interactive SQL is connected to (for example, SQL Anywhere 16.0.0.1403).
- **Host Name**  The network name of the computer the database server is running on.
- **Host TCP/IP Address**  The IP address of the computer the database server is running on.
- **Host Operating System**  The name and version number of the operating system used by the computer the database server is running on.
- **Server Name**  The name of the database server.
- **Server TCP/IP Port**  The port number used by the database server for the current connection.
- **Database Name**  The name of the database that Interactive SQL is connected to.
- **Database Character Set**  The character set used for CHAR columns in the database.
- **Connection String**  The connection string that was used to connect to the database or database server. Three asterisks replace passwords.

Properties that do not apply to the current connection are omitted. For example, if you connect to a database server using shared memory, then the TCP/IP port is omitted.

**Privileges**
None

**Side effects**
None

**See also**
- “Interactive SQL” [SQL Anywhere Server - Database Administration]

**Standards and compatibility**
- **SQL/2008**  Vendor extension.
Examples

Describe the columns in the Departments table:

```sql
DESCRIPT TABLE GROUPO.Departments;
```

Here is an example of the result set for this statement:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Nullable</th>
<th>Primary key</th>
</tr>
</thead>
<tbody>
<tr>
<td>DepartmentID</td>
<td>integer</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>DepartmentName</td>
<td>char(40)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DepartmentHeadID</td>
<td>integer</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

List the indexes for the Customers table:

```sql
DESCRIPT INDEX FOR TABLE GROUPO.Customers;
```

Here is an example of the results for this statement:

<table>
<thead>
<tr>
<th>Index Name</th>
<th>Columns</th>
<th>Unique</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX_customer_name</td>
<td>Surname,GivenName</td>
<td>0</td>
<td>Clustered</td>
</tr>
</tbody>
</table>

DETACH TRACING statement

Ends a diagnostic tracing session.

Syntax

```sql
DETACH TRACING { WITH | WITHOUT } SAVE
```

Parameters

<table>
<thead>
<tr>
<th>WITH SAVE clause</th>
<th>Specify WITH SAVE to save unsaved diagnostic data in the diagnostic tables.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WITHOUT SAVE clause</td>
<td>Specify WITHOUT SAVE if you do not want to save unsaved tracing data.</td>
</tr>
</tbody>
</table>

Remarks

Execute this statement from the database being profiled to stop sending diagnostic information to the diagnostic tables. If you specify the WITHOUT SAVE clause, you can still save the data later—assuming the tracing database is still running and another tracing session has not been started—by using the sa_save_trace_data system procedure.

To see the current tracing levels set for a database, look in the sa_diagnostic_tracing_level table.
Note
Tracing information is not unloaded as part of a database unload or reload operation. To transfer tracing information from one database to another, you must do so manually by copying the contents of the sa_diagnostic_* tables; however, this is not recommended.

Privileges
You must have the MANAGE PROFILING system privilege.

Side effects
None.

See also
● “ATTACH TRACING statement” on page 514
● “REFRESH TRACING LEVEL statement” on page 930
● “Diagnostic tracing” [SQL Anywhere Server - SQL Usage]
● “sa_diagnostic_tracing_level table” on page 1081
● “sa_save_trace_data system procedure” on page 1217

Standards and compatibility
● SQL/2008 Vendor extension.

DISCONNECT statement [ESQL] [Interactive SQL]
Drops a connection to a database.

Syntax
DISCONNECT [ connection-name | CURRENT | ALL ]

connection-name :
identifier
| string
| hostvar

Remarks
The DISCONNECT statement drops a connection with the database server and releases all resources used by it. If the connection to be dropped was named on the CONNECT statement, the name can be specified. Specifying ALL will drop all the application's connections to all database environments. CURRENT is the default, and drops the current connection.

Before closing the database connection, Interactive SQL automatically executes a COMMIT statement if the commit_on_exit option is set to On. If this option is set to Off, Interactive SQL performs an implicit ROLLBACK. By default, the commit_on_exit option is set to On.

This statement is not supported in procedures, triggers, events, or batches.
Privileges

None.

Side effects

None.

See also

- “DROP CONNECTION statement” on page 753
- “CONNECT statement [ESQL] [Interactive SQL]” on page 543
- “SET CONNECTION statement [Interactive SQL] [ESQL]” on page 966
- “Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 DISCONNECT comprises optional SQL language feature F771 of the SQL/2008 standard. The ability to specify DISCONNECT without a parameter is a vendor extension. The commit_on_exit option is a vendor extension.

Example

The following statement shows how to use DISCONNECT in embedded SQL:

```
EXEC SQL DISCONNECT :conn_name
```

The following statement shows how to use DISCONNECT from Interactive SQL to disconnect all connections:

```
DISCONNECT ALL;
```

**DROP CERTIFICATE statement**

Drops a certificate from the database.

**Syntax**

```
DROP CERTIFICATE certificate-name
```

**Remarks**

DROP CERTIFICATE deletes a certificate from the ISYSCERTIFICATE system table.

**Privileges**

You must have the MANAGE CERTIFICATES system privilege.

**Side effects**

Automatic commit.

**See also**

- “CREATE CERTIFICATE statement” on page 547
Standards and compatibility

- SQL/2008 Vendor extension.

Example

DROP CERTIFICATE mycert;

DROP CONNECTION statement

Drops a user’s connection to the database.

Syntax

DROP CONNECTION connection-id

Remarks

The DROP CONNECTION statement disconnects a user from the database by dropping the connection to the database.

The connection-id parameter is an integer constant. You can obtain the connection-id using the sa_conn_info system procedure.

This statement is not supported in procedures, triggers, events, or batches.

Privileges

You must have the DROP CONNECTION system privilege.

Side effects

None.

See also

- “CONNECT statement [ESQL] [Interactive SQL]” on page 543
- “sa_conn_info system procedure” on page 1106
- “Exception handlers” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following procedure drops a connection identified by its connection number. When executing the DROP CONNECTION statement from within a procedure, you should do so using the EXECUTE IMMEDIATE statement, as shown in this example:

```sql
CREATE PROCEDURE drop_connection_by_id( IN conn_number INTEGER )
BEGIN
    EXECUTE IMMEDIATE 'DROP CONNECTION ' || conn_number;
END;
```

The following statement drops the connection with ID number 4.
DROP DATABASE statement

Deletes all database files associated with a database.

Syntax

DROP DATABASE database-name [ KEY key ]

Remarks

The DROP DATABASE statement physically deletes all associated database files from disk. If the database file does not exist, or is not in a suitable condition for the database to be started, an error is generated.

DROP DATABASE cannot be used in stored procedures, triggers, events, or batches.

The database to be dropped must not be running when the DROP DATABASE statement is used. You cannot be connected to the database you are dropping. You must be connected to a different database. For example, connect to the utility database.

You must specify a key to drop a strongly encrypted database. The key can be either a string or a variable name.

This statement is not supported on Windows Mobile.

Privileges

Your ability to execute this statement depends on the setting for the -gu database option, and whether you have the SERVER OPERATOR system privilege.

Side effects

In addition to deleting the database files from disk, any associated transaction log file or transaction log mirror file is deleted.

See also

- “-gu database server option” [SQL Anywhere Server - Database Administration]
- “Erase utility (dberase)” [SQL Anywhere Server - Database Administration]
- “Deleting a Windows Mobile database using the device interface” [SQL Anywhere Server - Database Administration]
- “Connecting to the utility database” [SQL Anywhere Server - Database Administration]
- “CREATE DATABASE statement” on page 549
- “DatabaseKey (DBKEY) connection parameter” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

Execute the following statement to drop the database temp.db:
DROP DATABASE 'c:\temp\temp.db';

DROP DATATYPE statement

Removes a data type from the database.

Syntax

DROP DATATYPE datatype-name

Remarks

It is recommended that you use DROP DOMAIN rather than DROP DATATYPE, as DROP DOMAIN is the syntax used in the SQL/2008 standard. You cannot drop system-defined data types (such as MONEY or UNIQUEIDENTIFIERSTR) from a database.

Privileges

You must be the owner of the data type, or have the DROP DATATYPE or DROP ANY OBJECT system privilege.

Side effects

Automatic commit. Clears the Results tab in the Results pane in Interactive SQL.

See also

- “DROP DOMAIN statement” on page 756
- “CREATE DOMAIN statement” on page 562
- “ALTER DOMAIN statement” on page 435

Standards and compatibility

- SQL/2008  Domain support is optional SQL language feature F251 in the SQL/2008 standard. The DROP DATATYPE statement is a vendor extension.

Example

The following example creates and then drops a datatype called PhoneNum:

```
CREATE DATATYPE PhoneNum CHAR(12) NULL;
DROP DATATYPE PhoneNum;
```

DROP DBSPACE statement

Removes a dbspace from the database.

Syntax

DROP DBSPACE dbspace-name
Remarks

You must drop all tables in the dbspace before dropping the dbspace. You cannot use the DROP DBSPACE statement to drop the predefined dbspaces SYSTEM, TEMPORARY, TEMP, TRANSLOG, or TRANSLOGMIRROR.

DROP DBSPACE is prevented whenever the statement affects an object that is currently being used by another connection.

You must be the only connection to the database to execute this statement.

Privileges

You must have the MANAGE ANY DBSPACE system privilege.

Side effects

Automatic commit, and causes an implicit checkpoint. Clears the Results tab in the Results pane in Interactive SQL.

See also

- “CREATE DBSPACE statement” on page 557
- “ALTER DBSPACE statement” on page 432
- “Dropping a dbspace (Sybase Central)” [SQL Anywhere Server - Database Administration]
- “Dropping a dbspace (SQL)” [SQL Anywhere Server - Database Administration]
- “Predefined dbspaces” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

This example drops a fictitious dbspace, MyDBSpace, from the database.

```
DROP DBSPACE MyDBSpace;
```

DROP DOMAIN statement

Removes a domain (data type) from the database.

Syntax

```
DROP DOMAIN domain-name
```

Remarks

DROP DOMAIN is prevented if the data type is used in a table column, or in a procedure or function argument. You must change data types on all columns defined using the domain to drop the data type. It is recommended that you use DROP DOMAIN rather than DROP DATATYPE, as DROP DOMAIN is the syntax used in the SQL/2008 standard. You cannot drop system-defined data types (such as MONEY or UNIQUEIDENTIFIERSTR) from a database.
Privileges
You must be the owner of the domain, or have the DROP DATATYPE or DROP ANY OBJECT system privilege.

Side effects
Automatic commit. Clears the Results tab in the Results pane in Interactive SQL.

See also
● “CREATE DOMAIN statement” on page 562
● “ALTER DOMAIN statement” on page 435

Standards and compatibility
● SQL/2008 Domain support is optional SQL language feature F251 in the SQL/2008 standard.

Example
The following example creates and then drops the domain CustPhoneNumber.

    CREATE DOMAIN CustPhoneNumber CHAR(12) NULL;
    DROP DOMAIN CustPhoneNumber;

DROP EVENT statement
Drops an event from the database.

Syntax
    DROP EVENT [ IF EXISTS ] [ owner. ] event-name

Remarks
Use the IF EXISTS clause if you do not want an error returned when the DROP EVENT statement attempts to remove an event that does not exist.

Privileges
You must have either the MANAGE ANY EVENT or DROP ANY OBJECT system privilege.

Side effects
Automatic commit. Clears the Results tab in the Results pane in Interactive SQL.

See also
● “CREATE EVENT statement” on page 570
● “ALTER EVENT statement” on page 436
● “TRIGGER EVENT statement” on page 1018

Standards and compatibility
● SQL/2008 Vendor extension.
Example
   This example drops a fictitious example, MyEvent, from the database.

   DROP EVENT MyEvent;

**DROP EXTERNLOGIN statement**
   Drops an external login from the database.

**Syntax**

   DROP EXTERNLOGIN login-name TO remote-server

**Parameters**

   DROP clause      Specifies the local user login ID.

   TO clause        Specifies the name of the remote server. The local user's alternate login name and password
                     for that server is the external login that is deleted.

**Remarks**

   DROP EXTERNLOGIN deletes an external login from the database.

**Privileges**

   You must have the MANAGE ANY USER system privilege.

**Side effects**

   Automatic commit.

**See also**

   ● “CREATE EXTERNLOGIN statement” on page 578

**Standards and compatibility**

   ● SQL/2008      Vendor extension.

**Example**

   The following example drops the DBA external login to the fictitious remote server, sybase1.

   DROP EXTERNLOGIN DBA TO sybase1;

**DROP FUNCTION statement**

   Removes a function from the database.

**Syntax**

   DROP FUNCTION [ IF EXISTS ] [ owner.]function-name
Remarks
Use the IF EXISTS clause if you do not want an error returned when the DROP FUNCTION statement attempts to remove a function that does not exist.

DROP FUNCTION is prevented when the statement affects an object that is currently being used by another connection.

Privileges
You must be the owner of the function, or have the DROP ANY PROCEDURE or DROP ANY OBJECT system privilege.

Side effects
Automatic commit. Clears the Results tab in the Results pane in Interactive SQL.

See also
- “CREATE FUNCTION statement” on page 594
- “CREATE FUNCTION statement [External call]” on page 580
- “CREATE FUNCTION statement [Web service]” on page 586
- “ALTER FUNCTION statement” on page 439

Standards and compatibility
- SQL/2008 Core feature. The IF EXISTS clause is a vendor extension.

Example
This example drops the fictitious function, MyFunction, from the database.

    DROP FUNCTION MyFunction;

DROP INDEX statement
Removes an index from the database.

Syntax

    DROP INDEX [ IF EXISTS ] { [ [ owner.]table-name.]index-name | [ [ owner.]materialized-view-name. ]index-name }

Remarks
Use the IF EXISTS clause if you do not want an error returned when the DROP INDEX statement attempts to remove an index that does not exist.

When you specify the IF EXISTS clause and the named table cannot be located, an error is returned.

DROP INDEX is prevented when the statement affects an object that is currently being used by another connection.

The DROP INDEX statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.
Privileges
To drop an index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

To drop an index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

Side effects
Automatic commit. Clears the Results tab in the Results pane in Interactive SQL. The DROP INDEX statement closes all cursors for the current connection.

If you use the DROP INDEX statement to drop an index on a local temporary table an error is returned indicating that the index could not be found. Use the DROP TABLE statement to drop a local temporary table. Indexes on local temporary tables are dropped automatically when the local temporary table goes out of scope.

See also
- “CREATE INDEX statement” on page 599
- “ALTER INDEX statement” on page 441
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
This example drops a fictitious index, MyIndex, from the database.

    DROP INDEX MyIndex;

DROP LDAP SERVER statement
Drops an LDAP server configuration object.

Syntax

    DROP LDAP SERVER ldapua-server-name
    [ WITH DROP ALL REFERENCES ]
    [ WITH SUSPEND ]

Parameters

- WITH DROP ALL REFERENCES clause Specify the DROP ALL REFERENCES clause to drop an LDAP server configuration object that is referenced from a login policy.
WITH SUSPEND clause  Specify the WITH SUSPEND clause to drop an LDAP server configuration object that is in a READY or ACTIVE state.

Remarks
This statement removes the LDAP server configuration object from the SYSLDAPSERVER system view after checking that the LDAP server configuration object is not in the READY or ACTIVE state. The statement fails when the state is READY or ACTIVE to ensure that the LDAP server is not in active use. To override this check, specify the WITH SUSPEND clause.

By default, a reference to the LDAP server configuration object in a login policy also causes this statement to fail. To remove an LDAP server configuration object that is referenced in a login policy, add the DROP ALL REFERENCES clause. Adding DROP ALL REFERENCES does not remove the reference from the login policy; it allows you to drop the configuration object when there are references. You must still remove the reference to the LDAP server configuration object from the login policy.

Privileges
You must have the MANAGE ANY LDAP SERVER system privilege.

Side effects
Automatic commit.

See also
- “LDAP user authentication” [SQL Anywhere Server - SQL Usage]
- “CREATE LDAP SERVER statement” on page 603
- “ALTER LDAP SERVER statement” on page 442
- “VALIDATE LDAP SERVER statement” on page 1044
- “ALTER LOGIN POLICY statement” on page 445

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following example suspends an LDAP server configuration object named apps_primary that is referenced in a login policy, and then drops it.

```
DROP LDAP SERVER apps_primary WITH DROP ALL REFERENCES WITH SUSPEND;
```

DROP LOGIN POLICY statement
Drops a login policy.

Syntax
```
DROP LOGIN POLICY policy-name
```

Parameters

```
policy-name  The name of the login policy.
```
Remarks
The statement fails if you drop a policy that is assigned to a user. You cannot drop the root login policy.
Use the ALTER USER statement to change a user’s policy assignment.

Privileges
You must have the MANAGE ANY LOGIN POLICY system privilege.

Side effects
None.

See also
- “ALTER LOGIN POLICY statement” on page 445
- “ALTER USER statement” on page 508
- “COMMENT statement” on page 538
- “CREATE LOGIN POLICY statement” on page 607
- “CREATE USER statement” on page 721
- “DROP USER statement” on page 786
- “Login policies” [SQL Anywhere Server - Database Administration]
- “Deleting a login policy” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Examples
The following example creates a login policy, Test11, and then deletes it.

```
CREATE LOGIN POLICY Test11;
DROP LOGIN POLICY Test11;
```

### DROP MATERIALIZED VIEW statement

Removes a materialized view from the database.

**Syntax**

```
DROP MATERIALIZED VIEW [ IF EXISTS ] [ owner.]materialized-view-name
```

**Remarks**

All data in the table is automatically deleted as part of the dropping process. All indexes and keys for the materialized view are dropped as well.

Use the IF EXISTS clause if you do not want an error returned when the DROP MATERIALIZED VIEW statement attempts to remove a materialized view that does not exist.

You cannot execute a DROP MATERIALIZED VIEW statement on an object that is currently being used by another connection.
Executing a DROP MATERIALIZED VIEW statement changes the status of all dependent regular views to INVALID. To determine view dependencies before dropping a materialized view, use the sa_dependent_views system procedure.

**Privileges**
You must be the owner of the materialized view, or have the DROP ANY MATERIALIZED VIEW or DROP ANY OBJECT system privilege.

**Side effects**
Automatic commit. If the materialized view had been populated, DROP MATERIALIZED VIEW will trigger an automatic checkpoint. Clears the **Results** tab in the **Results** pane in Interactive SQL. Closes all cursors for the current connection.

When a view is dropped, all procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the fact that the view does not exist. The unloading and loading of procedures and triggers can affect performance if you are regularly dropping and creating views.

**See also**
- “CREATE MATERIALIZED VIEW statement” on page 612
- “ALTER MATERIALIZED VIEW statement” on page 450
- “REFRESH MATERIALIALIZED VIEW statement” on page 925
- “Advanced: Status and properties for materialized views” [SQL Anywhere Server - SQL Usage]
- “sa_dependent_views system procedure” on page 1123

**Standards and compatibility**
- SQL/2008 Vendor extension.

**Example**
The following example drops a fictitious materialized view, MyMaterializedView, from the database.

```
DROP MATERIALIZED VIEW MyMaterializedView;
```

**DROP MESSAGE statement**
Removes a message from the database.

**Syntax**
```
DROP MESSAGE msgnum
```

**Remarks**
None.

**Privileges**
You must be owner, or have the DROP MESSAGE or DROP ANY OBJECT system privilege.
Side effects

Automatic commit. Clears the Results tab in the Results pane in Interactive SQL.

See also

- “PRINT statement [T-SQL]” on page 918
- “CREATE MESSAGE statement [T-SQL]” on page 614
- “SYSUSERMESSAGE system view” on page 1414

Standards and compatibility

- SQL/2008 Vendor extension.
- Transact-SQL DROP MESSAGE supplies the functionality provided by the sp_dropmessage() system procedure in Adaptive Server Enterprise.

Example

The following example creates and then drops a new message. To run this example, you must also have the CREATE MESSAGE system privilege:

```
CREATE MESSAGE 20000 AS 'End of line reached';
DROP MESSAGE 20000;
```

DROP MIRROR SERVER statement

Note

Read-only scale-out and database mirroring each require a separate license. See “Separately licensed components” [SQL Anywhere 16 - Introduction].

Drops a mirror server.

Syntax

```
DROP MIRROR SERVER  mirror-server-name
```

Remarks

Removes the specified mirror server definition from the database.

The mirror database stops. If the mirror database is the only database running on the server, then the server also stops.

Privileges

You must have the MANAGE ANY MIRROR SERVER system privilege.

Side effects

Automatic commit.
See also

- “Database mirroring” [SQL Anywhere Server - Database Administration]
- “CREATE MIRROR SERVER statement” on page 615
- “ALTER MIRROR SERVER statement” on page 452
- “COMMENT statement” on page 538

Standards and compatibility

- SQL/2008 Vendor extension

Example

This example creates, and then drops, a mirror server named scaleout_primary2:

```sql
CREATE MIRROR SERVER "scaleout_primary2"
  AS PRIMARY
  connection_string =
  'server=scaleout_primary1;host=winxp-2:6871,winxp-3:6872';
DROP MIRROR SERVER "scaleout_primary2";
```

DROP PROCEDURE statement

Removes a procedure from the database.

Syntax

```sql
DROP PROCEDURE [ IF EXISTS ] [ owner.]procedure-name
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP PROCEDURE statement attempts to remove a procedure that does not exist.

You cannot execute a DROP PROCEDURE statement when the statement affects an object that is currently being used by another connection.

Privileges

You must be the owner of the procedure, or have the DROP ANY PROCEDURE or DROP ANY OBJECT system privilege.

Side effects

Automatic commit. Clears the Results tab in the Results pane in Interactive SQL.

See also

- “CREATE PROCEDURE statement” on page 639
- “CREATE PROCEDURE statement [External call]” on page 620
- “CREATE PROCEDURE statement [Web service]” on page 628
- “ALTER PROCEDURE statement” on page 456
Standards and compatibility

- **SQL/2008** Core feature. The IF EXISTS clause is a vendor extension.

Example

This example creates a procedure called NewDepartment, and then drops it. To run this example, you must also have the CREATE PROCEDURE privilege.

```sql
CREATE PROCEDURE NewDepartment(
    IN id INT,
    IN name CHAR(35),
    IN head_id INT )
BEGIN
    INSERT INTO GROUPO.Departments ( DepartmentID, DepartmentName, DepartmentHeadID )
    VALUES ( id, name, head_id );
END;
DROP PROCEDURE NewDepartment;
```

**DROP PUBLICATION statement [MobiLink] [SQL Remote]**

Drops a publication.

**Syntax**

```
DROP PUBLICATION [ IF EXISTS ] [ owner. ]publication-name
```

*owner, publication-name : identifier*

**Remarks**

This statement is applicable only to MobiLink and SQL Remote.

In MobiLink, a publication identifies synchronized data in a SQL Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

Use the IF EXISTS clause if you do not want an error returned when the DROP PUBLICATION statement attempts to remove a publication that does not exist.

DROP PUBLICATION requires exclusive access to all tables referred to in the publication.

**Privileges**

You must be the owner of the publication, or have the SYS_REPLICATION_ADMIN_ROLE system role.

**Side effects**

Automatic commit. All subscriptions to the publication are dropped.
See also
- “ALTER PUBLICATION statement [MobiLink] [SQL Remote]” on page 457
- “CREATE PUBLICATION statement [MobiLink] [SQL Remote]” on page 647
- SQL Anywhere MobiLink clients: “Publications” [MobiLink - Client Administration]
- UltraLite MobiLink clients: “DROP PUBLICATION statement [UltraLite]” [UltraLite - Database Management and Reference]
- SQL Anywhere MobiLink clients: “Dropping a publication” [MobiLink - Client Administration]
- SQL Remote: “Dropping a publication” [SQL Remote]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement drops the pub_contact publication.

```sql
DROP PUBLICATION pub_contact;
```

**DROP REMOTE MESSAGE TYPE statement [SQL Remote]**

Deletes a message type definition from a database.

**Syntax**

```sql
DROP REMOTE MESSAGE TYPE message-system
```

```sql
message-system :  
FILE  
| FTP  
| SMTP
```

**Remarks**

The statement removes a message type from a database.

**Privileges**

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

**Side effects**

Automatic commit.

**See also**

- “CREATE REMOTE [MESSAGE] TYPE statement [SQL Remote]” on page 651
- “SQL Remote message systems” [SQL Remote]
- “Deleting a message type” [SQL Remote]

**Standards and compatibility**

- SQL/2008 Vendor extension.
Example

The following statement drops the FILE message type from a database.

    DROP REMOTE MESSAGE TYPE FILE;

DROP REMOTE CONNECTION statement

Drops remote data access connections to a remote server.

Syntax

    DROP REMOTE CONNECTION TO server-name
    CLOSE { CURRENT | ALL | connection-id }
    [ FOR { EFFECTIVE USER | LOGIN USER | USER user-name } ]

Parameters

    server-name    The remote data access server that was specified in the CREATE SERVER statement.

    CLOSE clause   CLOSE CURRENT drops remote connections for the current local connection.
                    CLOSE ALL drops remote connections for all local connections.
                    CLOSE connection-id drops remote connections for the local connection with the specified ID.

    FOR clause     FOR EFFECTIVE USER drops remote connections that were created with the current effective user's externlogin credentials.
                    FOR LOGIN USER drops remote connections that were created with the current login user's externlogin credentials.
                    FOR USER user-name drops remote connections that were created with the externlogin credentials for user-name.

    If the FOR clause is omitted, then remote connections for all users are dropped.

Remarks

    The DROP REMOTE CONNECTION statement allows you to explicitly close connections to a remote server. You may find this useful when a remote connection becomes inactive or is no longer needed.

Privileges

    You must have the SERVER OPERATOR system privilege.

Side effects

    None

See also

    ● “CREATE SERVER statement” on page 657
    ● “ALTER SERVER statement” on page 463
Example

Drop all remote connections, whether they are the current connection or not, to the myServer server for the effective user:

```
DROP REMOTE CONNECTION TO myServer CLOSE ALL FOR EFFECTIVE USER;
```

Drop the remote connection to the myServer server for the current local connection for user2:

```
DROP REMOTE CONNECTION TO myServer CLOSE CURRENT FOR USER user2;
```

Drop all remote connections, whether they are the current connection or not, to the myServer server for user2:

```
DROP REMOTE CONNECTION TO myServer CLOSE ALL FOR USER user2;
```

Drop the remote connection to the myServer server for the current local connection with the current connection's login user:

```
DROP REMOTE CONNECTION TO myServer CLOSE CURRENT FOR LOGIN USER;
```

Drop the remote connection to the myServer server for the connection with the ID connection-id and the current effective user:

```
DROP REMOTE CONNECTION TO myServer CLOSE connectionId FOR EFFECTIVE USER;
```

**DROP ROLE statement**

Removes a role from the database, or converts a user-extended role back to a regular user.

**Syntax**

```
DROP ROLE [ FROM USER ] role-name  
 [ WITH { REVOKE | DROP OBJECTS } ]
```

**Parameters**

- `role-name` Specify the name of the role you are dropping or converting.
- FROM USER clause Specify this clause to convert a user-extended role back to a regular user. The user retains any login privileges, system privileges, and roles they had.
- WITH REVOKE clause Specify WITH REVOKE when there are other users who have been granted `role-name`.
- WITH DROP OBJECTS clause Specify WITH DROP OBJECTS to drop the objects owned by `role-name`. If any of the objects cannot be dropped, for example because the object is currently in use, then the statement returns an error. You cannot specify this clause if `role-name` is a user-extended role.

**Remarks**

A user-defined role can be dropped from the database, and a user-extended role can be converted back to a regular user, as long as all dependent roles meet the minimum required number of administrative users with active passwords, as set by the min_role_admin database option.
When you convert a user-extended role back to a regular user, ownership of objects remains with the user that is being converted back to a regular user.

When you convert a user-extended role back to a regular user, any privileges that were granted to role-name remain with the user after they have been converted.

If you convert a user-extended role back to a regular user and any other roles and/or users were granted the user-extended role, the WITH REVOKE clause must be specified or else the statement returns an error message and fails.

If any objects impacted by the drop operation are in use, the statement returns an error message and the statement fails.

Privileges
You must have administrative rights for the role being dropped.

Side effects
Automatic commit

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following statement converts a user-extended role named Joe back to a regular user. Objects owned by the user-extended role are now owned by the regular user, Joe. Users or roles that had been granted Joe retain the underlying privileges associated with the role.

    DROP ROLE FROM USER Joe;

The following statement drops a user-extended role named Jack from the database. If the role Jack owned any objects, ownership of the object reverts to user Jack. Users or roles that were granted Jack retains the underlying privileges associated with the role Jack.

    DROP ROLE Jack;

The following statement converts a user-extended role named Sam back to a regular user. Users and roles who had been granted Sam has the privileges of Jack revoked.

    DROP ROLE FROM USER Sam WITH REVOKE;

The following statement drops a role named Sales1. Users or roles that were granted Sales1 retain the underlying privileges associated with the Sales1.

    DROP ROLE Sales1;

The following statement drops a role named Sales2. Users or roles that had been granted Sales2 lose all underlying privileges associated with Sales2.

    DROP ROLE Sales2 WITH REVOKE;
The following statement converts a user-extended role named Marketing1 to a regular user named Marketing1, and drops any objects that it owned.

```
DROP ROLE FROM USER Marketing1 WITH DROP OBJECTS;
```

The following statement drops a role named Marketing2, drops the objects it owned, and revokes its underlying system privileges from those who had been granted the role.

```
DROP ROLE Marketing2 WITH REVOKE WITH DROP OBJECTS;
```

See also
- “ALTER ROLE statement” on page 460
- “CREATE ROLE statement” on page 652
- “min_role_admins option” [SQL Anywhere Server - Database Administration]
- “User-extended roles” [SQL Anywhere Server - Database Administration]

## DROP SEQUENCE statement

Drops a sequence.

**Syntax**

```
DROP SEQUENCE [ owner. ] sequence-name
```

**Remarks**

If the named sequence cannot be located, an error message is returned. When you drop a sequence, all synonyms for the name of the sequence are dropped automatically by the database server.

**Privileges**

You must be the owner of the sequence, or have the DROP ANY SEQUENCE or DROP ANY OBJECT system privilege.

**Side effects**

None

**See also**

- “Use of a sequence to generate unique values” [SQL Anywhere Server - SQL Usage]
- “ALTER SEQUENCE statement” on page 462
- “CREATE SEQUENCE statement” on page 655

**Standards and compatibility**


**Example**

The following example creates and then drops a sequence named Test:

```
CREATE SEQUENCE Test
START WITH 4
```
DROP SERVER statement

Drops a remote server from the SQL Anywhere catalog.

Syntax

DROP SERVER server-name

Remarks

DROP SERVER deletes a remote server from the SQL Anywhere catalogs. You must drop all the proxy tables that have been defined for the remote server before this statement will succeed.

This statement is not supported on Windows Mobile.

Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

Automatic commit.

See also

- “ALTER SERVER statement” on page 463
- “CREATE SERVER statement” on page 657
- “DROP REMOTE CONNECTION statement” on page 768
- “Remote data access” [SQL Anywhere Server - SQL Usage]
- “Remote servers” [SQL Anywhere Server - SQL Usage]
- “Directory access servers” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example drops a fictitious server named ase_prod:

```
DROP SERVER ase_prod;
```

DROP SERVICE statement

Drops a web service.
Syntax

**DROP SERVICE service-name**

Remarks

This statement deletes a web service listed in the ISYSWEBSERVICE system table.

Privileges

You must be the owner of the service, or have the MANAGE ANY WEB SERVICE system privilege.

Side effects

None.

See also

- “CREATE SERVICE statement [HTTP web service]” on page 662
- “CREATE SERVICE statement [SOAP web service]” on page 668
- “ALTER SERVICE statement [HTTP web service]” on page 466
- “ALTER SERVICE statement [SOAP web service]” on page 471
- “SYSWEBSERVICE system view” on page 1418

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following SQL statement drops a fictitious web service named WebServiceTable:

```
DROP SERVICE WebServiceTable;
```

**DROP SPATIAL REFERENCE SYSTEM statement**

Drops a spatial reference system.

Syntax

**DROP SPATIAL REFERENCE SYSTEM [ IF EXISTS ] name**

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP SPATIAL REFERENCE SYSTEM statement attempts to remove a spatial reference system that does not exist.

Privileges

You must be the owner, or have the MANAGE ANY SPATIAL OBJECT or DROP ANY OBJECT system privilege.

Side effects

None
See also

- “CREATE SPATIAL REFERENCE SYSTEM statement” on page 673
- “ALTER SPATIAL REFERENCE SYSTEM statement” on page 477
- “Spatial data” [SQL Anywhere Server - Spatial Data Support]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example drops a fictitious spatial reference system named Test.

```sql
DROP SPATIAL REFERENCE SYSTEM Test;
```

**DROP SPATIAL UNIT OF MEASURE statement**

Drops a spatial unit of measurement.

Syntax

```sql
DROP SPATIAL UNIT OF MEASURE [ IF EXISTS ] identifier
```

Remarks

Use the IF EXISTS clause if you do not want an error returned when the DROP SPATIAL UNIT OF MEASURE statement attempts to remove a spatial unit of measure that does not exist.

Privileges

You must be the owner of the spatial unit of measure, or have the MANAGE ANY SPATIAL OBJECT or DROP ANY OBJECT system privilege.

Side effects

None

See also

- “CREATE SPATIAL UNIT OF MEASURE statement” on page 681
- “Spatial data” [SQL Anywhere Server - Spatial Data Support]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example drops a fictitious spatial unit of measure named Test.

```sql
DROP SPATIAL UNIT OF MEASURE Test;
```
DROP STATEMENT statement [ESQL]

Frees statement resources.

Syntax

```
DROP STATEMENT [ owner.]statement-name
```

```
statement-name : identifier
| hostvar
```

Remarks

The DROP STATEMENT statement frees resources used by the named prepared statement. These resources are allocated by a successful PREPARE statement, and are normally not freed until the database connection is released.

To drop the statement, you must first have prepared the statement.

Privileges

None.

Side effects

None.

See also

- “PREPARE statement [ESQL]” on page 915

Standards and compatibility

- SQL/2008 Vendor extension. In the SQL/2008 standard, this functionality is provided by the DEALLOCATE PREPARE statement, which is part of the optional SQL language feature B032, "Extended dynamic SQL".

Example

The following are examples of DROP STATEMENT use:

```
EXEC SQL DROP STATEMENT S1;
EXEC SQL DROP STATEMENT :stmt;
```

DROP STATISTICS statement

Erases all column statistics on the specified columns.

Syntax

```
DROP STATISTICS [ ON ] [ owner.]object-name [ ( column-list ) ]
```

```
object-name :
table-name
```
Remarks
The SQL Anywhere optimizer uses column statistics to determine the best strategy for executing each statement. SQL Anywhere automatically gathers and updates these statistics. Column statistics are stored permanently in the database in the ISYSCOLSTAT system table. Column statistics gathered while processing one statement are available when searching for efficient ways to execute subsequent statements.

Occasionally, the column statistics can become inaccurate or relevant statistics may be unavailable. This condition is most likely to arise when few queries have been executed since a large amount of data was added, updated, or deleted.

The DROP STATISTICS statement deletes all internal statistical data from the ISYSCOLSTAT system table for the specified columns. This drastic step leaves the optimizer with no access to essential statistical information. Without these statistics, the optimizer can generate inefficient data access plans, causing poor database performance.

The DROP STATISTICS statement requires an exclusive lock on the table against which it is being performed. Execution of the statement cannot proceed until all other connections that refer to the table have either committed or rolled back the referring transactions, or closed any open cursors that refer to the table.

This statement should be used only during problem determination or when reloading data into a database that differs substantially from the original data.

Privileges
You must be the table owner, or have the MANAGE ANY STATISTICS or DROP ANY OBJECT system privilege.

Side effects
Automatic commit.

See also
- “CREATE STATISTICS statement” on page 682
- “Optimizer estimates and statistics” [SQL Anywhere Server - SQL Usage]
- “SYSCOLSTAT system view” on page 1351

Standards and compatibility
- SQL/2008 Vendor extension.

DROP SUBSCRIPTION statement [SQL Remote]
Drops a subscription for a user from a publication.
Syntax

```
DROP SUBSCRIPTION TO publication-name [ ( subscription-value ) ] FOR subscriber-id, ...
```

```
subscription-value  :  string
subscriber-id       :  string
```

Parameters

```
publication-name    The name of the publication to which the user is being subscribed. This can include the owner of the publication.
```

```
subscription-value    A string that is compared to the subscription expression of the publication. This value is required because a user can have more than one subscription to a publication.
```

```
subscriber-id        The user ID of the subscriber to the publication.
```

Remarks

Drops a SQL Remote subscription for a user ID to a publication in the current database. The user ID will no longer receive updates when data in the publication is changed.

In SQL Remote, publications and subscriptions are two-way relationships. If you drop a subscription for a remote user to a publication on a consolidated database, you should also drop the subscription for the consolidated database on the remote database to prevent updates on the remote database being sent to the consolidated database.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

See also

- “CREATE SUBSCRIPTION statement [SQL Remote]” on page 683
- “STOP SUBSCRIPTION statement [SQL Remote]” on page 1008
- “SYSSUBSCRIPTION system view” on page 1395

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement drops a subscription for the SamS user ID to the pub_contact publication.

```
DROP SUBSCRIPTION TO pub_contact
FOR Sam_Singer;
```
DROP SYNCHRONIZATION PROFILE statement [MobiLink]

Deletes a SQL Anywhere synchronization profile.

Syntax

```
DROP SYNCHRONIZATION PROFILE [ IF EXISTS ] name
```

Parameters

`name`  The name of the synchronization profile to delete.

Remarks

Synchronization profiles are named collections of synchronization options that can be used to control synchronization. Use the IF EXISTS clause if you do not want an error returned when the DROP SYNCHRONIZATION PROFILE statement attempts to remove a synchronization profile that does not exist.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

See also

- “CREATE SYNCHRONIZATION PROFILE statement [MobiLink]” on page 685
- “ALTER SYNCHRONIZATION PROFILE statement [MobiLink]” on page 481

Standards and compatibility

- SQL/2008  Vendor extension.

DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]

Drops a synchronization subscription in a remote database.

Syntax

```
DROP SYNCHRONIZATION SUBSCRIPTION { subscription-name |
  TO publication-name
  [ FOR ml-username, ... ] }
```

Parameters

`subscription-name`  Specifies the name of the subscription to drop.

`TO` clause  Specifies the name of a publication.

`FOR` clause  Specifies one more users.
Omitting this clause drops the default settings for the publication.

Remarks
Requires exclusive access to all tables referred to in the publication.

Privileges
You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects
Automatic commit.

See also
- “ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” on page 482
- “CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” on page 686
- “SYSSYNC system view” on page 1396
- “Dropping MobiLink subscriptions” [MobiLink - Client Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Examples
The following example drops the subscription named sales:

```sql
DROP SYNCHRONIZATION SUBSCRIPTION sales;
```

The following example drops the subscription between the MobiLink user SSinger and the publication called sales_publication:

```sql
DROP SYNCHRONIZATION SUBSCRIPTION
TO user.sales_publication
FOR "SSinger";
```

The following example omits the FOR clause, and so drops the default settings for the publication called sales_publication:

```sql
DROP SYNCHRONIZATION SUBSCRIPTION
TO user.sales_publication;
```

**DROP SYNCHRONIZATION USER statement [MobiLink]**

Drops one or more synchronization users from a SQL Anywhere remote database.

Syntax

```
DROP SYNCHRONIZATION USER ml-username, ...
```

*ml-username* : identifier
Remarks
Drop one or more synchronization users from a MobiLink remote database.

You must have exclusive access to all tables referred to by publications subscribed to by the user.

Privileges
You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects
All subscriptions associated with the user are also deleted.

See also
- “ALTER SYNCHRONIZATION USER statement [MobiLink]” on page 485
- “CREATE SYNCHRONIZATION USER statement [MobiLink]” on page 688
- “SYSSYNC system view” on page 1396

Standards and compatibility
- SQL/2008 Vendor extension.

Example
Remove MobiLink user SSinger from the database.

    DROP SYNCHRONIZATION USER SSinger;

DROP TABLE statement
Removes a table from the database.

Syntax
    DROP TABLE [ IF EXISTS ] [ owner.]table-name

Remarks
When you remove a table, all data in the table is automatically deleted as part of the dropping process. All indexes and keys for the table are dropped as well.

Use the IF EXISTS clause if you do not want an error returned when the DROP TABLE statement attempts to remove a table that does not exist.

You cannot execute a DROP TABLE statement when the statement affects a table that is currently being used by another connection. Execution of a DROP TABLE statement is also prevented if there is a materialized view dependent on the table.

When you execute a DROP TABLE statement, the status of all dependent regular views change to INVALID. To determine view dependencies before dropping a table, use the sa_dependent_views system procedure.
Global temporary tables cannot be dropped unless all users that have referenced the temporary table have disconnected.

**Privileges**

You must be the owner of the table, or have the DROP ANY TABLE or DROP ANY OBJECT system privilege.

**Side effects**

Automatic commit. DROP TABLE may also cause an automatic checkpoint. Clears the **Results** tab in the **Results** pane in Interactive SQL. Executing a DROP TABLE statement closes all cursors for the current connection.

You can use the DROP TABLE statement to drop a local temporary table.

**See also**

- “Dropping a table” [SQL Anywhere Server - SQL Usage]
- “CREATE TABLE statement” on page 690
- “ALTER TABLE statement” on page 486
- “sa_dependent_views system procedure” on page 1123

**Standards and compatibility**

- **SQL/2008** DROP TABLE is a core feature of the SQL/2008 standard. The IF EXISTS clause is a vendor extension. The ability to drop a declared local temporary table with the DROP TABLE statement is a vendor extension.

**Example**

This example drops the fictitious MyTable table from the database.

    DROP TABLE MyTable;

This example drops the fictitious MyTable table from the database. Because IF EXISTS is specified, if the table does not exist, an error is not returned.

    DROP TABLE IF EXISTS MyTable;

---

**DROP TEXT CONFIGURATION statement**

Drops a text configuration object.

**Syntax**

```
DROP TEXT CONFIGURATION [ owner.]text-config-name
```

**Remarks**

Attempting to drop a text configuration object with dependent text indexes results in an error. You must drop the dependent text indexes before dropping the text configuration object.

Text configuration objects are stored in the `ISYSTEXTCONFIG` system table.
Privileges

You must be the owner of the text configuration object, or have the DROP ANY TEXT CONFIGURATION or DROP ANY OBJECT system privilege.

Side effects

Automatic commit

See also

- “DROP TEXT INDEX statement” on page 782
- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Text configuration object concepts and reference” [SQL Anywhere Server - SQL Usage]
- “SYSTEXTCONFIG system view” on page 1405
- “CREATE TEXT CONFIGURATION statement” on page 709
- “ALTER TEXT CONFIGURATION statement” on page 501

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statements create and drop the mytextconfig text configuration object:

```
CREATE TEXT CONFIGURATION mytextconfig FROM default_char;
DROP TEXT CONFIGURATION mytextconfig;
```

DROP TEXT INDEX statement

Removes a text index from the database.

Syntax

```
DROP TEXT INDEX text-index-name
ON [owner.]table-name
```

Parameters

- **ON clause**  Use this clause to specify the table or materialized view on which the text index was built.

Remarks

You must drop dependent text indexes before you can drop a text configuration object.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.
Privileges

To drop a text index on a table, you must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

To drop a text index on a materialized view, you must be the owner of the materialized view, or have one of the following privileges:

- DROP ANY INDEX system privilege
- DROP ANY OBJECT system privilege

Side effects

Automatic commit

See also

- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Text index concepts and reference” [SQL Anywhere Server - SQL Usage]
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]
- “SYSTEXTCONFIG system view” on page 1405
- “CREATE TEXT INDEX statement” on page 711
- “ALTER TEXT INDEX statement” on page 504
- “DROP TEXT INDEX statement” on page 782
- “REFRESH TEXT INDEX statement” on page 928
- “TRUNCATE TEXT INDEX statement” on page 1021

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statements create and drop the TextIdx text index:

```
CREATE TEXT INDEX TextIdx ON GROUPO.MarketingInformation ( Description )
DROP TEXT INDEX TextIdx ON GROUPO.MarketingInformation;
```

DROP TRACE EVENT statement

Drops a user-defined trace event.

Syntax

```
DROP TRACE EVENT [ IF EXISTS ] trace-event-name
```

Remarks

This statement only drops user-defined trace events. If you do not want an error returned when the DROP TRACE EVENT statement attempts to remove a trace event that does not exist, use the IF EXISTS
clause. If one or more event tracing sessions reference the trace event, then the trace event cannot be dropped until all the referencing trace sessions are dropped.

**System privileges**

You must have the MANAGE ANY TRACE SESSION system privilege.

**Side effects**

None

**See also**

- “Event tracing” [SQL Anywhere Server - Database Administration]
- “CREATE TEMPORARY TRACE EVENT statement” on page 705
- “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
- “ALTER TRACE EVENT SESSION statement” on page 506
- “DROP TRACE EVENT SESSION statement” on page 784
- “NOTIFY TRACE EVENT statement” on page 903
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_fields system procedure” on page 1312
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_session_events system procedure” on page 1314
- “sp_trace_event_target_options system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

Drop the trace event named my_event:

```
DROP TRACE EVENT my_event;
```

**DROP TRACE EVENT SESSION statement**

Drops a trace event session.

**Syntax**

```
DROP TRACE EVENT SESSION [ IF EXISTS ] session-name UNCONDITIONALLY
```

**Remarks**

When UNCONDITIONALLY is specified, dropping an active session automatically stops the session before removing its definition. Otherwise, an error is returned.
System privileges
You must have the MANAGE ANY TRACE SESSION system privilege.

Side effects
None

See also
- “Event tracing” [SQL Anywhere Server - Database Administration]
- “CREATE TEMPORARY TRACE EVENT statement” on page 705
- “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
- “ALTER TRACE EVENT SESSION statement” on page 506
- “DROP TRACE EVENT SESSION statement” on page 784
- “NOTIFY TRACE EVENT statement” on page 903
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_fields system procedure” on page 1312
- “sp_trace_event_sessions system procedure” on page 1317
- “sp_trace_event_session_events system procedure” on page 1314
- “sp_trace_event_session_targets system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement drops the trace event session named my_session:

    DROP TRACE EVENT SESSION my_session;

DROP TRIGGER statement
Removes a trigger from the database.

Syntax
    DROP TRIGGER [ IF EXISTS ] [ owner.] [ table-name.]trigger-name

Remarks
Use the IF EXISTS clause if you do not want an error returned when the DROP statement attempts to remove a database object that does not exist.

Privileges
To drop a trigger on a table, one of the following must be true:
You are the owner of the table.

- You have ALTER privilege on the table.
- You have the ALTER ANY TABLE system privilege.
- You have the ALTER ANY OBJECT system privilege.

To drop a trigger on a view owned by someone else, you must have either the ALTER ANY VIEW or ALTER ANY OBJECT system privilege.

**Side effects**

Automatic commit. Clears the Results tab in the Results pane in Interactive SQL.

**See also**

- “CREATE TRIGGER statement” on page 713
- “ALTER TRIGGER statement” on page 507
- “ROLLBACK TRIGGER statement” on page 952
- “Dropping a trigger” [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**

- **SQL/2008** DROP TRIGGER comprises part of optional SQL language feature T211, "Basic trigger capability", of the SQL/2008 standard. The IF EXISTS clause is a vendor extension.

**Example**

This example creates, and then drops, a trigger called emp_upper_postal_code to ensure that postal codes are in upper case before updating the Employees table. If the trigger does not exist, an error is returned.

```
CREATE TRIGGER emp_upper_postal_code
BEFORE UPDATE OF PostalCode
ON GROUPO.Employees
REFERENCING NEW AS new_emp
FOR EACH ROW
WHEN ( ISNUMERIC( new_emp.PostalCode ) = 0 )
BEGIN
    -- Ensure postal code is uppercase (employee might be
    -- in Canada where postal codes contain letters)
    SET new_emp.PostalCode = UPPER(new_emp.PostalCode)
END;
DROP TRIGGER MyTrigger;
```

**DROP USER statement**

Drops a user.

**Syntax**

```
DROP USER user-name
```

**Parameters**

- **user-name** The name of the user you are dropping.
Privileges

You must have the MANAGE ANY USER system privilege.

Remarks

Dropping a user also deletes all database objects (such as tables or procedures) that they own, as well as any external logins for the user. In addition, if the user is specified in the USER clause of any services, then those services are also dropped.

The user being removed cannot be connected to the database when the statement is executed.

Side effects

None.

See also

- “ALTER LOGIN POLICY statement” on page 445
- “ALTER USER statement” on page 508
- “COMMENT statement” on page 538
- “CREATE LOGIN POLICY statement” on page 607
- “CREATE USER statement” on page 721
- “DROP LOGIN POLICY statement” on page 761
- “Login policies” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example creates and then drops the user SQLTester.

```
CREATE USER SQLTester IDENTIFIED BY pass1234;
DROP USER SQLTester;
```

DROP VARIABLE statement

Eliminates a SQL variable.

Syntax

```
DROP VARIABLE [ IF EXISTS ] identifier
```

Remarks

The DROP VARIABLE statement eliminates a SQL variable that was previously created using the CREATE VARIABLE statement. Variables are automatically eliminated when the database connection is released. Variables are often used for large objects, so eliminating them after use or setting them to NULL can free up significant resources (primarily disk space).

Use the IF EXISTS clause if you do not want an error returned when the DROP statement attempts to remove a database object that does not exist.
Privileges

None.

Side effects

None.

See also

- “CREATE VARIABLE statement” on page 722
- “SET statement” on page 985

Standards and compatibility

- SQL/2008 Vendor extension.

**DROP VIEW statement**

Removes a view from the database.

**Syntax**

```
DROP VIEW [ IF EXISTS ] [ owner. ] view-name
```

**Remarks**

Use the IF EXISTS clause if you do not want an error returned when the DROP VIEW statement attempts to remove a view that does not exist.

When you execute the DROP VIEW statement, the status of all dependent regular views change to INVALID. To determine view dependencies before dropping a view, use the sa_dependent_views system procedure.

If you execute a DROP VIEW statement on a view that has one or more INSTEAD OF triggers, an error is returned. You must drop the trigger before the view can be dropped or altered.

**Privileges**

You must be the owner of the view, or have the DROP ANY VIEW or DROP ANY OBJECT system privilege.

**Side effects**

Automatic commit. Clears the Results tab in the Results pane in Interactive SQL. Executing a DROP VIEW statement closes all cursors for the current connection.

When a view is dropped, all procedures and triggers are unloaded from memory, so that any procedure or trigger that references the view reflects the fact that the view does not exist. The unloading and loading of procedures and triggers can affect performance if you are regularly dropping and creating views.
See also

- “CREATE VIEW statement” on page 724
- “ALTER VIEW statement” on page 511
- “sa_dependent_views system procedure” on page 1123

Standards and compatibility

- **SQL/2008** DROP VIEW is a core feature of the SQL/2008 standard. The IF EXISTS clause is a vendor extension.

**Example**

The following example creates a view called MyView, and then drops it. You must be able to select from the Employees table to execute the CREATE VIEW statement in the example.

```sql
CREATE VIEW MyView
    AS SELECT * FROM GROUPO.Employees;
DROP VIEW MyView;
```

**EXCEPT statement**

Returns the set difference of two query blocks.

**Syntax**

```sql
[ WITH temporary-views ] main-query-block
EXCEPT [ ALL | DISTINCT ] except-query-block
[ ORDER BY [ integer | select-list-expression-name ] [ ASC | DESC ], ... ]
[ FOR XML xml-mode ]
[ OPTION( query-hint, ... ) ]
```

- **query-hint**: MATERIALIZED VIEW OPTIMIZATION option-value
- **force_optimization**: option-name = option-value

- **main-query-block**: query-block. See “Common elements in SQL syntax” on page 421.

- **except-query-block**: query-block. See “Common elements in SQL syntax” on page 421.

- **option-name**: identifier

- **option-value**: hostvar (indicator allowed)
  - string
  - identifier
  - number

**Parameters**

- **main-query-block**: A query block, comprising a SELECT statement or a query expression (possibly nested).
**except-query-block** A query block, comprising a SELECT statement or a query expression (possibly nested).

**FOR XML clause** For information about the FOR XML clause, see “SELECT statement” on page 955.

**OPTION clause** Use this clause to specify hints for executing the statement. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION *option-value*
- FORCE OPTIMIZATION
- *option-name = option-value*. A OPTION( isolation_level = ... ) specification in the query text overrides all other means of specifying isolation level for a query.

**Remarks**

The EXCEPT statement returns all rows in main-query-block except those that also appear in the except-query-block. Specify EXCEPT or EXCEPT DISTINCT if you do not want duplicates from main-query-block to appear as duplicates in the result. Otherwise, specify EXCEPT ALL. Query blocks can be nested.

The use of EXCEPT alone is equivalent to EXCEPT DISTINCT.

The main-query-block and the except-query-block must be UNION-compatible; they must each have the same number of items in their respective SELECT lists, and the types of each expression should be comparable. If corresponding items in two SELECT lists have different data types, SQL Anywhere chooses a data type for the corresponding column in the result and automatically convert the columns in each query-block appropriately.

EXCEPT ALL implements bag difference rather than set difference. For example, if main-query-block contains 5 (duplicate) rows with specific values, and except-query-block contains 2 duplicate rows with identical values, then EXCEPT ALL will return 3 rows.

The results of EXCEPT are the same as the results of EXCEPT ALL if main-query-block does not contain duplicate rows.

The column names displayed are the same column names that are displayed for the first query-block and these names are used to determine the expression names to be matched with the ORDER BY clause. An alternative way of customizing result set column names is to use a common table expression (the WITH clause).

**Privileges**

You must own the tables referenced in query-block, or have the SELECT ANY TABLE privilege.

**Side effects**

None
See also

- “INTERSECT statement” on page 870
- “UNION statement” on page 1025
- “SELECT statement” on page 955
- “OPTION clause, SELECT statement” on page 962

Standards and compatibility

- **SQL/2008**  EXCEPT DISTINCT is a core feature of the SQL/2008 standard; EXCEPT ALL comprises the optional SQL language feature F304. Explicitly specifying the DISTINCT keyword with EXCEPT is optional SQL language feature T551 of the SQL/2008 standard. Specifying an ORDER BY clause with EXCEPT is SQL language feature F850. A query-block that contains an ORDER BY clause constitutes SQL/2008 feature F851. A query block that contains a row-limit clause (SELECT TOP or LIMIT) comprises optional SQL language feature F857 or F858, depending on the context. The FORXML clause and the OPTION clause are vendor extensions.

- **Transact-SQL**  EXCEPT is not supported by Adaptive Server Enterprise. However, both EXCEPT ALL and EXCEPT DISTINCT can be used in the Transact-SQL dialect supported by SQL Anywhere.

Example

For examples of EXCEPT usage, see “Set operators and NULL” [SQL Anywhere Server - SQL Usage].

EXECUTE IMMEDIATE statement [SP]

Enables dynamically constructed statements to be executed from within a procedure.

Syntax 1

EXECUTE IMMEDIATE [ execute-option ] string-expression

execute-option :
   WITH QUOTES [ ON | OFF ]
   | WITH ESCAPES { ON | OFF }
   | WITH BATCH { ON | OFF }
   | WITH RESULT SET { ON | OFF }

Syntax 2 - Transact-SQL

EXECUTE ( string-expression )

Parameters

**WITH QUOTES clause**  When you specify WITH QUOTES or WITH QUOTES ON, any double quotes in the string-expression are assumed to delimit an identifier. When you do not specify WITH QUOTES, or specify WITH QUOTES OFF, the treatment of double quotes in the string-expression depends on the current setting of the quoted_identifier option.

WITH QUOTES is useful when an object name that is passed into the stored procedure is used to construct the statement that is to be executed, but the name might require double quotes and the procedure might be called when the quoted_identifier option is set to Off.
**WITH ESCAPES clause**  
WITH ESCAPES OFF causes any escape sequences (such as \n, \x, or \\) in the string-expression to be ignored. For example, two consecutive backslashes remain as two backslashes, rather than being converted to a single backslash. The default setting is equivalent to WITH ESCAPES ON.

One use of WITH ESCAPES OFF is for easier execution of dynamically constructed statements referencing file names that contain backslashes.

In some contexts, escape sequences in the string-expression are transformed before the EXECUTE IMMEDIATE statement is executed. For example, compound statements are parsed before being executed, and escape sequences are transformed during this parsing, regardless of the WITH ESCAPES setting. In these contexts, WITH ESCAPES OFF prevents further translations from occurring. For example:

```sql
BEGIN
    DECLARE String1 LONG VARCHAR;
    DECLARE String2 LONG VARCHAR;
    EXECUTE IMMEDIATE 'SET String1 = ''One backslash: \\\
    ''; 
    EXECUTE IMMEDIATE WITH ESCAPES OFF 'SET String2 = ''Two backslashes: \\
    ''; 
    SELECT String1, String2
END
```

**WITH BATCH clause**  
The WITH BATCH clause allows you to control the execution of batches in EXECUTE IMMEDIATE statements. Setting WITH BATCH OFF provides protection against inadvertent SQL-injection when the procedure is run.

WITH BATCH ON is the default, except for procedures owned by dbo.

When WITH BATCH OFF is used, the statement specified by string-expression must be a single statement.

**WITH RESULT SET clause**  
The WITH RESULT SET clause allows the server to define correctly the procedure containing it. Specifying WITH RESULT SET ON or WITH RESULT SET OFF affects both what happens when the procedure is created, as well as what happens when the procedure is executed. The default option is WITH RESULT SET OFF.

You can have an EXECUTE IMMEDIATE statement return a result set by specifying WITH RESULT SET ON. With this clause, the containing procedure is marked as returning a result set.

**Remarks**

The EXECUTE IMMEDIATE statement extends the range of statements that can be executed from within procedures and triggers. It lets you execute dynamically prepared statements, such as statements that are constructed using the parameters passed in to a procedure.

For information about the supported syntax for named parameters, see “Named parameters” on page 87.

Literal strings in the statement must be enclosed in single quotes. String literals cannot span multiple lines.

Only global variables can be referenced in a statement executed by EXECUTE IMMEDIATE.
Only Syntax 2 can be used inside Transact-SQL stored procedures and triggers.

Statements executed with EXECUTE IMMEDIATE do not have their plans cached.

Privileges

None.

Side effects

None. However, if the statement is a data definition statement with an automatic commit as a side effect, that commit does take place.

See also

- “Named parameters” on page 87
- “quoted_identifier option” [SQL Anywhere Server - Database Administration]
- “EXECUTE IMMEDIATE used in procedures, triggers, user-defined functions, and batches” [SQL Anywhere Server - SQL Usage]
- “CREATE PROCEDURE statement” on page 639
- “CREATE PROCEDURE statement [External call]” on page 620
- “CREATE PROCEDURE statement [Web service]” on page 628
- “BEGIN statement” on page 523
- “EXECUTE statement [ESQL]” on page 794
- “EXECUTE statement [T-SQL]” on page 796

Standards and compatibility

- **SQL/2008** EXECUTE IMMEDIATE is optional SQL language feature B031, "Basic dynamic SQL", of the SQL/2008 standard. The execute-option syntax is a vendor extension. The SQL/2008 standard prohibits the use of EXECUTE IMMEDIATE that returns a result set.

- **Transact-SQL** Syntax 2 is the Transact-SQL dialect's syntax for EXECUTE IMMEDIATE. The execute-option syntax is not supported by Adaptive Server Enterprise.

Examples

The following procedure creates a table, where the table name is supplied as a parameter to the procedure.

```sql
CREATE PROCEDURE CreateTableProc(
    IN tablename char(30)
)
BEGIN
    EXECUTE IMMEDIATE
    'CREATE TABLE ' || tablename ||
    ' ( column1 INT PRIMARY KEY)'
END;
```

To call the procedure and create a table called mytable:

```sql
CALL CreateTableProc( 'mytable' );
```
EXECUTE statement [ESQL]

Executes a prepared SQL statement.

Syntax 1

```
EXECUTE statement
[ USING { hostvar-list | [ SQL ] DESCRIPTOR sqlda-name } ]
[ INTO { into-hostvar-list | [ SQL ] DESCRIPTOR into-sqlda-name } ]
[ ARRAY :row-count ]
```

row-count : integer | hostvar

statement : identifier | hostvar | string

sqlda-name : identifier

into-sqlda-name : identifier

Syntax 2

```
EXECUTE IMMEDIATE statement
```

statement : string | hostvar

Parameters

**USING clause**  Results from a SELECT statement or a CALL statement are put into either the variables in the variable list or the program data areas described by the named SQLDA. The correspondence is one-to-one from the OUTPUT (selection list or parameters) to either the host variable list or the SQLDA descriptor array.

**INTO clause**  If EXECUTE INTO is used with an INSERT statement, the inserted row is returned in the second descriptor. For example, when using auto-increment primary keys or BEFORE INSERT triggers that generate primary key values, the EXECUTE statement provides a mechanism to re-fetch the row immediately and determine the primary key value that was assigned to the row. The same thing can be achieved by using @@identity with auto-increment keys.

**ARRAY clause**  The optional ARRAY clause can be used with prepared INSERT statements to allow wide inserts, which insert more than one row at a time and which can improve performance. The integer value is the number of rows to be inserted. The SQLDA must contain a variable for each entry (number of rows * number of columns). The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on.

Remarks

The EXECUTE statement can be used for any SQL statement that can be prepared. Cursors are used for SELECT statements or CALL statements that return many rows from the database.

After successful execution of an INSERT, UPDATE or DELETE statement, the sqlerrd[2] field of the SQLCA (SQLCOUNT) is filled in with the number of rows affected by the operation.

- **Syntax 1**  Execute the named dynamic statement, which was previously prepared. If the dynamic statement contains host variable placeholders that supply information for the request (bind variables),
either the sqlda-name must specify a C variable which is a pointer to a SQLDA containing enough descriptors for all the bind variables occurring in the statement, or the bind variables must be supplied in the hostvar-list.

- **Syntax 2** A short form to PREPARE and EXECUTE a statement that does not contain bind variables or output. The SQL statement contained in the string or host variable is immediately executed, and is dropped on completion.

**Privileges**

The required privileges depend on the statement being executed.

**Side effects**

None.

**See also**

- “Named parameters” on page 87
- “EXECUTE IMMEDIATE statement [SP]” on page 791
- “PREPARE statement [ESQL]” on page 915
- “DECLARE CURSOR statement [ESQL] [SP]” on page 729
- “Cursors in embedded SQL” [SQL Anywhere Server - Programming]

**Standards and compatibility**

- **SQL/2008** The EXECUTE statement comprises part of optional SQL language feature B031, "Basic dynamic SQL", of the SQL/2008 standard. The INTO clause is part of optional language feature B032, "Extended dynamic SQL". The ARRAY clause is a vendor extension.

The EXECUTE IMMEDIATE statement supported with embedded SQL is also part of optional SQL language feature B031.

**Example**

This example executes a DELETE statement.

```
EXEC SQL EXECUTE IMMEDIATE
'DELETE FROM Employees WHERE EmployeeID = 105';
```

This example executes a prepared DELETE statement.

```
EXEC SQL PREPARE del_stmt FROM
'DELETE FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE del_stmt USING :employee_number;
```

This example executes a prepared query.

```
EXEC SQL PREPARE sel1 FROM
'SELECT Surname FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE sel1 USING :employee_number INTO :surname;
```
EXECUTE statement [T-SQL]

Invokes a procedure (an Adaptive Server Enterprise-compatible alternative to the CALL statement), executes a prepared SQL statement in Transact-SQL.

Syntax 1 - calling a stored procedure

```
[ EXECUTE | EXEC ][ @return_status = ] [creator.]procedure_name [ argument, ... ]
```

argument :

```
[ @parameter-name = ] expression
[ [ @parameter-name = ] @variable [ output ]
```

Syntax 2 - executing dynamic statements within T-SQL stored procedures and triggers

```
EXECUTE ( string-expression )
```

Remarks

Syntax 1 is implemented for Transact-SQL compatibility. EXECUTE calls a stored procedure, optionally supplying procedure parameters and retrieving output values and return status information. In Watcom SQL, use the CALL or EXECUTE IMMEDIATE statements.

With Syntax 2, you can execute dynamic statements within Transact-SQL stored procedures and triggers. The EXECUTE statement extends the range of statements that can be executed from within procedures and triggers. It lets you execute dynamically prepared statements, such as statements that are constructed using the parameters passed in to a procedure. Literal strings in the statement must be enclosed in single quotes, and the statement must be on a single line. Syntax 2 of the EXECUTE statement is implemented for Transact-SQL compatibility, but can be used in either Transact-SQL or Watcom SQL batches and procedures.

The Transact-SQL EXECUTE statement does not have a way to signify that a result set is expected. One way to indicate that a Transact-SQL procedure returns a result set is to include something like the following:

```
IF 1 = 0
  SELECT 1 AS a
```

You can also execute statements within Transact-SQL stored procedures and triggers.

For information about the supported syntax for named parameters, see “Named parameters” on page 87.

Privileges

When calling a procedure, you must be the owner of the procedure, or have the EXECUTE ANY PROCEDURE system privilege.

When executing dynamic statements within T-SQL stored procedures and triggers, the required privileges depend on the statement being executed.

Side effects

None.
See also

- “Named parameters” on page 87
- “CALL statement” on page 530
- “EXECUTE statement [ESQL]” on page 794
- “EXECUTE IMMEDIATE statement [SP]” on page 791

Standards and compatibility

- **SQL/2008**  Syntax 1 is a vendor extension. Syntax 2 offers functionality equivalent to the EXECUTE IMMEDIATE statement in the SQL/2008 standard, which is optional SQL language feature B031, "Basic dynamic SQL". However, the syntax of Syntax 2 differs from that of the SQL/2008 standard.

Example

The following procedure illustrates Syntax 1.

```sql
CREATE PROCEDURE p1( @var INTEGER = 54 ) 
AS 
PRINT 'on input @var = %1!', @var
DECLARE @intvar integer
SELECT @intvar=123
SELECT @var=@intvar
PRINT 'on exit @var = %1!', @var;
```

The following statement executes the procedure, supplying the input value of 23 for the parameter. If you are connected from an Open Client or JDBC application, the PRINT messages are displayed in the client window. If you are connected from an ODBC or embedded SQL application, the messages are displayed in the database server messages window.

```sql
EXECUTE p1 23;
```

The following is an alternative way of executing the procedure, which is useful if there are several parameters.

```sql
EXECUTE p1 @var = 23;
```

The following statement executes the procedure, using the default value for the parameter

```sql
EXECUTE p1;
```

The following statement executes the procedure, and stores the return value in a variable for checking return status.

```sql
EXECUTE @status = p1 23;
```

**EXIT statement [Interactive SQL]**

Leaves Interactive SQL.

Syntax

```sql
{ EXIT | QUIT | BYE } [ return-code ]

return-code : number | connection-variable
```
Remarks
This statement closes the Interactive SQL window if you are running Interactive SQL as a windowed program, or terminates Interactive SQL altogether when running in command-prompt (batch) mode. In both cases, the database connection is also closed. Before closing the database connection, Interactive SQL automatically executes a COMMIT statement if the commit_on_exit option is set to On. If this option is set to Off, Interactive SQL performs an implicit ROLLBACK. By default, the commit_on_exit option is set to On.

The optional return code can be checked in batch files to determine the success or failure of the statements in an Interactive SQL script file. The default return code is 0.

Privileges
None.

Side effects
This statement automatically performs a commit if option commit_on_exit is set to On (the default); otherwise it performs an implicit rollback.

On Windows operating systems the optional return value is available as ERRORLEVEL.

See also
● “SET OPTION statement” on page 972
● “Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility
● SQL/2008 Vendor extension.

Examples
The following example sets the Interactive SQL return value to 1 if there are any rows in table T, or to 0 if T contains no rows.

    CREATE VARIABLE rowCount INT;
    CREATE VARIABLE retcode INT;
    SELECT COUNT(*) INTO rowCount FROM GROUPO.Products;
    IF( rowCount > 0 ) THEN
        SET retcode = 1;
    ELSE
        SET retcode = 0;
    END IF;
    EXIT retcode;
Note
You cannot write the following statement because EXIT is an Interactive SQL statement (not a SQL
statement), and you cannot include any Interactive SQL statement in other SQL block statements.

```
CREATE VARIABLE rowCount INT;
SELECT COUNT(*) INTO rowCount FROM T;
IF ( rowCount > 0 ) THEN
    EXIT 1    //  <-- not allowed
ELSE
    EXIT 0    //  <-- not allowed
END IF;
```

EXPLAIN statement [ESQL]
Retrieves a text specification of the optimization strategy used for a particular cursor.

**Syntax**

```
EXPLAIN PLAN FOR CURSOR cursor-name
{   INTO hostvar | USING DESCRIPTOR sqlda-name }
```

cursor-name : identifier | hostvar

sqlda-name : identifier

**Remarks**

The EXPLAIN statement retrieves a text representation of the optimization strategy for the named cursor. 
The cursor must be previously declared and opened.

The hostvar or sqlda-name variable must be of string type. The optimization string specifies in what order
the tables are searched, and also which indexes are being used for the searches if any.

This string may be long, depending on the query, and has the following format:

```
table (index), table (index), ...
```

If a table has been given a correlation name, the correlation name will appear instead of the table name. 
The order that the table names appear in the list is the order in which they are accessed by the database 
server. After each table is a parenthesized index name. This is the index that is used to access the table. If 
no index is used (the table is scanned sequentially) the letters "seq" will appear for the index name. If a 
particular SQL SELECT statement involves subqueries, a colon (:) will separate each subquery's 
optimization string. These subquery sections will appear in the order that the database server executes the 
queries.

After successful execution of the EXPLAIN statement, the sqlerrd field of the SQLCA 
(SQLOIOESTIMATE) is filled in with an estimate of the number of input/output operations required to 
fetch all rows of the query.

A discussion with quite a few examples of the optimization string can be found in “Performance 
monitoring and diagnostic tools” [SQL Anywhere Server - SQL Usage].
You can only execute this statement on cursors you opened.

Privileges
None.

Side effects
None.

See also
- “DECLARE CURSOR statement [ESQL] [SP]” on page 729
- “PREPARE statement [ESQL]” on page 915
- “FETCH statement [ESQL] [SP]” on page 800
- “CLOSE statement [ESQL] [SP]” on page 537
- “OPEN statement [ESQL] [SP]” on page 904
- “Cursors in embedded SQL” [SQL Anywhere Server - Programming]
- “The SQL Communication Area (SQLCA)” [SQL Anywhere Server - Programming]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example illustrates the use of EXPLAIN:

```sql
EXEC SQL BEGIN DECLARE SECTION;
  char plan[300];
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE employee_cursor CURSOR FOR
  SELECT EmployeeID, Surname
  FROM Employees
  WHERE Surname like :pattern;
EXEC SQL OPEN employee_cursor;
EXEC SQL EXPLAIN PLAN FOR CURSOR employee_cursor INTO :plan;
printf( "Optimization Strategy: '%s'.n", plan );
```

The plan variable contains the following string:

'Employees <seq>'

**FETCH statement [ESQL] [SP]**

Positions, or re-positions, a cursor to a specific row, and then copies expression values from that row into variables accessible from within the stored procedure or application.

**Syntax 1 [SP]**

```
FETCH [ cursor-position ] cursor-name
INTO variable-list [ FOR UPDATE ]
```
Syntax 2 [ESQL]

```
FETCH [ cursor-position ] cursor-name
   [ INTO { hostvar-list } ] [ USING [ SQL ] DESCRIPTOR sqlda-name ]
   [ PURGE ]
   [ BLOCK n ]
   [ FOR UPDATE ]
   [ ARRAY fetch-count ]
```

cursor-position :
   NEXT | PRIOR | FIRST | LAST
   | { ABSOLUTE | RELATIVE } row-count

row-count : number | hostvar

cursor-name : identifier | hostvar

hostvar-list : may contain indicator variables

variable-list : stored procedure variables

sqlda-name : identifier

fetch-count : integer | hostvar

Parameters

**INTO clause**  The INTO clause is optional. If it is not specified, the FETCH statement positions the cursor only. The hostvar-list is for embedded SQL use only.

**cursor position**  An optional positional parameter allows the cursor to be moved before a row is fetched. If not specified, NEXT is assumed. If the fetch includes a positioning parameter and the position is outside the allowable cursor positions, the SQLWARNING NOTFOUND warning is issued and the SQLCOUNT field indicates the offset from a valid position.

The OPEN statement initially positions the cursor before the first row.

**NEXT clause**  Next is the default positioning, and causes the cursor to advance one row before the row is fetched.

**PRIOR clause**  Causes the cursor to back up one row before fetching.

**RELATIVE clause**  RELATIVE positioning is used to move the cursor by a specified number of rows in either direction before fetching. A positive number indicates moving forward and a negative number indicates moving backward. So, a NEXT is equivalent to RELATIVE 1 and PRIOR is equivalent to RELATIVE -1. RELATIVE 0 retrieves the same row as the last fetch statement on this cursor.

**ABSOLUTE clause**  The ABSOLUTE positioning parameter is used to go to a particular row. A zero indicates the position before the first row.

A one (1) indicates the first row, and so on. Negative numbers are used to specify an absolute position from the end of the cursor. A negative one (-1) indicates the last row of the cursor.

**FIRST clause**  A short form for ABSOLUTE 1.
**LAST clause**  A short form for ABSOLUTE -1.

**Note**
Inserts and some updates to DYNAMIC SCROLL cursors can cause problems with cursor positioning. The database server does not put inserted rows at a predictable position within a cursor unless there is an ORDER BY clause on the SELECT statement. Sometimes the inserted row does not appear until the cursor is closed and opened again.

This behavior occurs if a temporary table had to be created to open the cursor.

The UPDATE statement can cause a row to move in the cursor. This will happen if the cursor has an ORDER BY that uses an existing index (a temporary table is not created).

**BLOCK clause**  Rows may be fetched by the client application more than one at a time. This is referred to as block fetching, prefetching, or multi-row fetching. The first fetch causes several rows to be sent back from the database server. The client buffers these rows, and subsequent fetches are retrieved from these buffers without a new request to the database server.

The BLOCK clause is for use in embedded SQL only. It gives the client and server a hint about how many rows may be fetched by the application. The special value of 0 means the request is sent to the database server and a single row is returned (no row blocking). The BLOCK clause will reduce the number of rows included in the next prefetch to the BLOCK value. To increase the number of rows prefetched, use the PrefetchRows connection parameter.

If you do not specify a BLOCK clause, the value specified on OPEN is used.

FETCH RELATIVE 0 always re-fetches the row.

If prefetch is disabled for the cursor, the BLOCK clause is ignored and rows are fetched one at a time. If ARRAY is also specified, then the number of rows specified by ARRAY are fetched.

**PURGE clause**  The PURGE clause is for use in embedded SQL only. It causes the client to flush its buffers of all rows, and then send the fetch request to the database server. This fetch request may return a block of rows.

**FOR UPDATE clause**  The FOR UPDATE clause indicates that the fetched row will subsequently be updated with an UPDATE WHERE CURRENT OF CURSOR statement. This clause causes the database server to put an intent lock on the row. The lock is held until the end of the current transaction.

**ARRAY clause**  The ARRAY clause is for use in embedded SQL only. It allows so-called wide fetches, which retrieve more than one row at a time, and which may improve performance.

To use wide fetches in embedded SQL, include the fetch statement in your code as follows:

```
EXEC SQL FETCH ... ARRAY nnn
```

where ARRAY *nnn* is the last item of the FETCH statement. The fetch count *nnn* can be a host variable. The SQLDA must contain *nnn* * (columns per row) variables. The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on.
Remarks

The FETCH statement retrieves one row from the named cursor. The cursor must have been previously opened.

- **Embedded SQL use**  The embedded SQL FETCH statement does not support arrays.

A DECLARE CURSOR statement must appear before the FETCH statement in the C source code, and the OPEN statement must be executed before the FETCH statement. If a host variable is being used for the cursor name, the DECLARE statement actually generates code and must be executed before the FETCH statement.

The server returns in SQLCOUNT the number of records fetched, and always returns a SQLCOUNT greater than zero unless there is an error or warning.

If the SQLSTATE_NOTFOUND warning is returned on the fetch, the sqlerrd[2] field of the SQLCA (SQLCOUNT) contains the number of rows by which the attempted fetch exceeded the allowable cursor positions. The value is 0 if the row was not found but the position is valid; for example, executing FETCH RELATIVE 1 when positioned on the last row of a cursor. The value is positive if the attempted fetch was beyond the end of the cursor, and negative if the attempted fetch was before the beginning of the cursor. The cursor is positioned on the last row if the attempted fetch was beyond the end of the cursor, and on the first row if the attempted fetch was before the beginning of the cursor.

After successful execution of the fetch statement, the sqlerrd[1] field of the SQLCA (SQLIOCOUNT) is incremented by the number of input/output operations required to perform the fetch. This field is actually incremented on every database statement.

- **Single row fetch**  One row from the result of the SELECT statement is put into the variables in the variable list. The correspondence is one-to-one from the SELECT list to the host variable list.

- **Multi-row fetch**  One or more rows from the result of the SELECT statement are put into either the variables in variable-list or the program data areas described by sqlda-name. In either case, the correspondence is one-to-one from the SELECT list to either the hostvar-list or the sqlda-name descriptor array.

Privileges

The cursor must be opened and you must have SELECT privilege on the tables, or be owner of the tables referenced in the declaration of the cursor, or have the SELECT ANY TABLE system privilege.

Side effects

A FETCH statement may cause multiple rows to be retrieved from the server to the client if prefetching is enabled.
Standards and compatibility

- **SQL/2008** With minor exceptions, Syntax 1 of the FETCH statement is a core feature of the SQL/2008 standard. Scrolling options other than NEXT constitute optional SQL language feature F431, "Read-only scrollable cursors". SQL Anywhere does not support the optional FROM clause of the FETCH statement as documented in the SQL/2008 standard.

Syntax 2 is a vendor extension.

The FOR UPDATE, PURGE, ARRAY, BLOCK, and USING [SQL] DESCRIPTOR clauses are vendor extensions.

Example

The following is an embedded SQL example:

```sql
EXEC SQL DECLARE cur_employee CURSOR FOR
SELECT EmployeeID, Surname FROM Employees;
EXEC SQL OPEN cur_employee;
EXEC SQL FETCH cur_employee
INTO :emp_number, :emp_name:indicator;
```

The following is a procedure example:

```sql
BEGIN
    DECLARE cur_employee CURSOR FOR
    SELECT Surname
    FROM Employees;
    DECLARE name CHAR(40);
    OPEN cur_employee;
    lp: LOOP
        FETCH NEXT cur_employee into name;
        IF SQLCODE <> 0 THEN LEAVE lp END IF;
        ...
    END LOOP;
    CLOSE cur_employee;
END
```
FOR statement

Repeats the execution of a statement list once for each row in a cursor.

Syntax

[ statement-label : ]
FOR for-loop-name AS cursor-name [ cursor-type ] CURSOR
{ FOR statement [ FOR \{ UPDATE \{ cursor-concurrency \} | READ ONLY \}] | USING variable-name }
DO statement-list
END FOR [ statement-label ]

cursor-type:

| NO SCROLL
| DYNAMIC SCROLL
| SCROLL
| INSENSITIVE
| SENSITIVE

cursor-concurrency:

| BY \{ VALUES | TIMESTAMP | LOCK \}

variable-name:
identifier

Parameters

NO SCROLL clause  A cursor declared NO SCROLL is restricted to moving forward through the result set using FETCH NEXT and FETCH RELATIVE 0 seek operations.

As rows cannot be returned to once the cursor leaves the row, there are no sensitivity restrictions on the cursor. When a NO SCROLL cursor is requested, SQL Anywhere supplies the most efficient kind of cursor, which is an asensitive cursor.

DYNAMIC SCROLL clause  DYNAMIC SCROLL is the default cursor type. DYNAMIC SCROLL cursors can use all formats of the FETCH statement.

When a DYNAMIC SCROLL cursor is requested, SQL Anywhere supplies an asensitive cursor. When using cursors there is always a trade-off between efficiency and consistency. Asensitive cursors provide efficient performance at the expense of consistency.

SCROLL clause  A cursor declared SCROLL can use all formats of the FETCH statement. When a SCROLL cursor is requested, SQL Anywhere supplies a value-sensitive cursor.

SQL Anywhere must execute value-sensitive cursors in such a way that result set membership is guaranteed. DYNAMIC SCROLL cursors are more efficient and should be used unless the consistent behavior of SCROLL cursors is required.

INSENSITIVE clause  A cursor declared INSENSITIVE has its values and membership fixed over its lifetime. The result set of the SELECT statement is materialized when the cursor is opened. FETCHING from an INSENSITIVE cursor does not see the effect of any other INSERT, UPDATE, MERGE, PUT, or DELETE statement from any connection, including the connection that opened the cursor.

SENSITIVE clause  A cursor declared SENSITIVE is sensitive to changes to membership or values of the result set.
FOR UPDATE clause FOR UPDATE is the default. Cursors default to FOR UPDATE for single-table queries without an ORDER BY clause, or if the ansi_update_constraints option is set to Off. When the ansi_update_constraints option is set to Cursors or Strict, then cursors over a query containing an ORDER BY clause default to READ ONLY. However, you can explicitly mark cursors as updatable using the FOR UPDATE clause.

FOR READ ONLY clause A cursor declared FOR READ ONLY cannot be used in UPDATE (positioned), DELETE (positioned), or PUT statements. Because it is expensive to allow updates over cursors with an ORDER BY clause or a join, cursors over a query containing a join of two or more tables are READ ONLY and cannot be made updatable unless the ansi_update_constraints database option is Off. In response to any request for a cursor that specifies FOR UPDATE, SQL Anywhere provides either a value-sensitive cursor or a sensitive cursor. Insensitive and asensitive cursors are not updatable.

See also
- “Sensitive cursors” [SQL Anywhere Server - Programming]
- “Insensitive cursors” [SQL Anywhere Server - Programming]
- “Value-sensitive cursors” [SQL Anywhere Server - Programming]
- “Asensitive cursors” [SQL Anywhere Server - Programming]

Remarks
The FOR statement is a control statement that allows you to execute a list of SQL statements once for each row in a cursor. The FOR statement is equivalent to a compound statement with a DECLARE for the cursor and a DECLARE of a variable for each column in the result set of the cursor followed by a loop that fetches one row from the cursor into the local variables and executes statement-list once for each row in the cursor.

Valid cursor types include dynamic scroll (default), scroll, no scroll, sensitive, and insensitive.

The name and data type of each local variable is derived from the statement used in the cursor. With a SELECT statement, the data types are the data types of the expressions in the SELECT list. The names are the SELECT list item aliases, if they exist; otherwise, they are the names of the columns. Any SELECT list item that is not a simple column reference must have an alias. With a CALL statement, the names and data types are taken from the RESULT clause in the procedure definition.

The LEAVE statement can be used to resume execution at the first statement after the END FOR. If the ending statement-label is specified, it must match the beginning statement-label.

The cursor created by a FOR statement is implicitly opened WITH HOLD, so statements executed within the loop that cause a COMMIT do not cause the cursor to be closed.

Caution
If you do not specify cursor-name, cursor-type is used as cursor-name.

Privileges
None.

Side effects
None.
Standards and compatibility

- **SQL/2008** The FOR statement is part of optional SQL/2008 language feature P002, "Computational completeness". The USING clause of the FOR statement is a vendor extension. As with the DECLARE CURSOR statement, the use of cursor-concurrency is a vendor extension, as are the combinations of cursor sensitivity and cursor scrollability options.

Example

The following fragment illustrates the use of the FOR loop.

```sql
FOR names AS curs INSENSITIVE CURSOR FOR
SELECT Surname
FROM Employees
DO
  CALL search_for_name( Surname );
END FOR;
```

This fragment also illustrates the use of the FOR loop.

```sql
BEGIN
  FOR names AS curs SCROLL CURSOR FOR
  SELECT EmployeeID, GivenName FROM Employees where EmployeeID < 130
  FOR UPDATE BY VALUES
  DO
    MESSAGE 'emp: ' || GivenName;
  END FOR;
END
```

The following example shows the FOR loop being used inside of a procedure called myproc, which returns the top 10 employees from the Employees table, depending on the sort order specified when calling the procedure (asc for ascending, and desc for descending).

```sql
CALL sa_make_object( 'procedure', 'myproc' ) ;
ALTER PROCEDURE myproc ( IN @order_by VARCHAR(20) DEFAULT NULL )
RESULT ( Surname person_name_t )
BEGIN
  DECLARE @sql LONG VARCHAR;
  DECLARE @msg LONG VARCHAR;
  DECLARE LOCAL TEMPORARY TABLE temp_names( surnames person_name_t );
  SET @sql = 'SELECT TOP(10) * FROM Employees AS t ' ;
  CASE @order_by
  WHEN 'asc' THEN
    SET @sql = @sql || 'ORDER BY t.Surname ASC';
    SET @msg = 'Sorted ascending by last name: ' ;
  WHEN 'desc' THEN
    SET @sql = @sql || 'ORDER BY t.Surname DESC';
    SET @msg = 'Sorted ascending by last name: ' ;
  END CASE;
```
FOR loop_name AS curs SCROLL CURSOR USING @sql
DO
  INSERT INTO temp_names( surnames ) VALUES( Surname );
  MESSAGE( @msg || Surname );
END FOR;
SELECT * FROM temp_names;
END ;

Calling the myproc procedure and specifying asc (for example, CALL myproc( 'asc' );) returns the following results:

<table>
<thead>
<tr>
<th>Surname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahmed</td>
</tr>
<tr>
<td>Barker</td>
</tr>
<tr>
<td>Barletta</td>
</tr>
<tr>
<td>Bertrand</td>
</tr>
<tr>
<td>Bigelow</td>
</tr>
<tr>
<td>Blaikie</td>
</tr>
<tr>
<td>Braun</td>
</tr>
<tr>
<td>Breault</td>
</tr>
<tr>
<td>Bucceri</td>
</tr>
<tr>
<td>Butterfield</td>
</tr>
</tbody>
</table>

FORWARD TO statement

Sends native syntax SQL statements to a remote server.

Syntax 1

FORWARD TO server-name sql-statement

Syntax 2

FORWARD TO [ server-name ]

Parameters

server-name  The name of the remote server.

sql-statement A command in the native SQL syntax of the remote server. The command or group of commands is enclosed in braces ({}) or single quotes.
 Remarks

The FORWARD TO statement enables users to specify the server to which a passthrough connection is required. The statement can be used in two ways:

● **Syntax 1**  Send a single statement to a remote server.

● **Syntax 2**  Place SQL Anywhere into passthrough mode for sending a series of statements to a remote server. All subsequent statements are passed directly to the remote server. To turn passthrough mode off, execute FORWARD TO without a `server-name` specification.

If you encounter an error from the remote server while in passthrough mode, you must still execute a FORWARD TO statement to turn passthrough off.

When establishing a connection to `server-name` on behalf of the user, the database server uses one of the following:

● A remote login alias set using CREATE EXTERNLOGIN

● If a remote login alias is not set up, the name and password used to communicate with SQL Anywhere

If the connection cannot be made to the server specified, the reason is contained in a message returned to the user.

After statements are passed to the requested server, any results are converted into a form that can be recognized by the client program.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>The FORWARD TO statement is a server directive and cannot be used in stored procedures, triggers, events, or batches.</td>
</tr>
</tbody>
</table>

 Privileges

None

 Side effects

The remote connection is set to AUTOCOMMIT (unchained) mode for the duration of the FORWARD TO session. Any work that was pending before the FORWARD TO statement is automatically committed.

 See also

● “PASSTHROUGH statement [SQL Remote]” on page 913

 Example

The following example sends a SQL statement to the remote server RemoteASE:

```sql
FORWARD TO RemoteASE { SELECT * FROM titles ;}
```

The following example shows a passthrough session with the remote server aseprod:

```
FORWARD TO aseprod;
   SELECT * FROM titles;
```
SQL statements

```
SELECT * FROM authors;
FORWARD TO;
```

Standards and compatibility
- SQL/2008 Vendor extension.

FROM clause
Specifies the database tables or views involved in a DELETE, SELECT, or UPDATE statement. When used within a SELECT statement, the FROM clause can also be used in a MERGE or INSERT statement.

Syntax
```
FROM table-expression, ...
```

```
table-expression :
  table-name
  view-name
  procedure-name
  derived-table
  lateral-derived-table
  join-expression
  ( table-expression, ... )
  openstring-expression
  apply-expression
  contains-expression
  dml-derived-table
```

```
table-name :
  [ userid.]table-name
  [ AS ] correlation-name
  [ WITH ( hint [...] ) ]
  [ FORCE INDEX ( index-name ) ]
```

```
view-name :
  [ userid.]view-name [ [ AS ] correlation-name ]
  [ WITH ( table-hint ) ]
```

```
procedure-name :
  [ owner.]procedure-name ( [ parameter, ... ] )
  [ WITH ( column-name data-type, ... ) ]
  [ AS ] correlation-name
```

```
derived-table :
  ( select-statement )
  [ AS ] correlation-name [ ( column-name, ... ) ]
```

```
lateral-derived-table :
  LATERAL ( select-statement | table-expression )
  [ AS ] correlation-name [ ( column-name, ... ) ]
```

```
join-expression :
  table-expression join-operator table-expression
  [ ON join-condition ]
```
join-operator:
  [ KEY | NATURAL ] [ join-type ] JOIN
  | CROSS JOIN

join-type:
  INNER
  | LEFT [ OUTER ]
  | RIGHT [ OUTER ]
  | FULL [ OUTER ]

hint:
  table-hint | index-hint

table-hint:
  READPAST
  | UPDLOCK
  | XLOCK
  | FASTFIRSTRROW
  | HOLDLOCK
  | NOLOCK
  | READDOWNCOMMITTED
  | REPEATAKERead
  | SERIALIZABLE

index-hint:
  NO INDEX
  | INDEX ( [ PRIMARY KEY | FOREIGN KEY ] index-name [, ...] ) [ INDEX ONLY { ON | OFF } ]
  | CLUSTERED INDEX [ INDEX ONLY { ON | OFF } ]

openstring-expression:
  OPENSTRING ( { FILE | VALUE } string-expression )
  WITH ( rowset-schema )
  [ OPTION ( scan-option ... ) ]
  [ AS ] correlation-name

apply-expression:
  table-expression { CROSS | OUTER } APPLY table-expression

contains-expression:
  { table-name | view-name } CONTAINS ( column-name [, ...], contains-query ) [ ] [ AS ] score-correlation-name

rowset-schema:
  column-schema-list
  | TABLE [owner.]table-name [ ( column-list ) ]

column-schema-list:
  { column-name user-or-base-type | filler( ) } [ , ... ]

column-list:
  { column-name | filler( ) } [ , ... ]

scan-option:
  BYTE ORDER MARK { ON | OFF }
  | COMMENTS INTRODUCED BY comment-prefix
Parameters

- **table-name**  A base table or temporary table. Tables owned by a different user can be qualified by specifying the user ID. Tables owned by user-defined roles that the user is a grantee of are found by default without specifying the user ID.

- **view-name**  Specifies a view to include in the query. As with tables, views owned by a different user can be qualified by specifying the user ID. Views owned by groups to which the current user belongs are found by default without specifying the user ID. Although the syntax permits table hints on views, these hints have no effect.

- **procedure-name**  A stored procedure that returns a result set. This clause applies to the FROM clause of SELECT statements only. The parentheses following the procedure name are required even if the procedure does not take parameters. DEFAULT can be specified in place of an optional parameter.

The argument list can be specified by position or by using keyword format. By position, the arguments match up with the corresponding parameter in the parameter list for the procedure (DEFAULT can be used for an optional parameter). By keyword, the arguments are matched up with the named parameters. For information about the supported syntax for named parameters, see “Named parameters” on page 87.

If the stored procedure returns multiple result sets, only the first is used.
The WITH clause provides a way of specifying column name aliases for the procedure result set. If a
WITH clause is specified, the number of columns must match the number of columns in the procedure
result set, and the data types must be compatible with those in the procedure result set. If no WITH
clause is specified, the column names and types are those defined by the procedure definition. The
following query illustrates the use of the WITH clause:

```
SELECT sp.ident, sp.quantity, Products.name
FROM GROUPO.ShowCustomerProducts( 149 )
  WITH ( ident INT, description CHAR(20), quantity INT ) sp
JOIN GROUPO.Products
ON sp.ident = Products.ID;
```

For embedded SQL applications, when you create a procedure without a RESULT clause and the
procedure returns a variable result set, a DESCRIBE of the SELECT statement referencing the
procedure may fail. To prevent the failure of the DESCRIBE, it is recommended that you include a
WITH clause that describes the expected result set schema.

Selecting from a procedure generates a temporary table. For example, the following query creates a
local temporary table:

```
BEGIN
... my: LOOP
  BEGIN
    SELECT TOP 1 NUMBER INTO conn_id FROM sa_conn_info( ) ORDER BY
    NUMBER DESC;
  END;
... ENDD LOOP my;
END
```

- **derived-table** You can supply a SELECT statement instead of table or view name in the FROM
  clause. A SELECT statement used in this way is called a derived table, and it must be given an alias.
  For example, the following statement contains a derived table, MyDerivedTable, which ranks
  products in the Products table by UnitPrice.

```
SELECT TOP 3 *
FROM ( SELECT Description, 
    Quantity, 
    UnitPrice, 
    RANK() OVER ( ORDER BY UnitPrice ASC ) 
    AS Rank
    FROM GROUPO.Products ) AS MyDerivedTable
ORDER BY Rank;
```

- **lateral-derived-table** A derived table, stored procedure, or joined table that may include
  references to objects in the parent statement (outer references). You must use a lateral derived table to
  use an outer reference in the FROM clause.

You can use outer references only to tables that precede the lateral derived table in the FROM clause.
For example, you cannot use an outer reference to an item in the SELECT list.

The table and the outer reference must be separated by a comma. For example, the following queries
are valid:

```
SELECT *
FROM A, LATERAL( B LEFT OUTER JOIN C ON ( A.x = B.x ) ) myLateralDT;
```
SELECT *
FROM A, LATERAL( SELECT * FROM B WHERE A.x = B.x ) myLateralDT;

SELECT *
FROM A, LATERAL( procedure-name( A.x ) ) myLateralDT;

Specifying LATERAL (table-expression) is equivalent to specifying LATERAL (SELECT * FROM table-expression).

- openstring-expression Specify an OPENSTRING clause to query within a file or a BLOB, treating the content of these sources as a set of rows. When doing so, you also specify information about the schema of the file or BLOB for the result set to be generated, since you are not querying a defined structure such as a table or view. This clause applies to the FROM clause of a SELECT statement. It is not supported for UPDATE or DELETE statements.

The ROWID function is supported over the result set of a table generated by an OPENSTRING expression.

The following subclauses and parameters of the OPENSTRING clause are used to define and query data within files and BLOBs:

**FILE and VALUE clauses** Use the FILE clause to specify the file to query. Use the VALUE clause to specify the BLOB expression to query. The data type for the BLOB expression is assumed to be LONG BINARY. You can specify the READ_CLIENT_FILE function as a value to the VALUE clause.

```
Note
If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].
```

If neither the FILE nor VALUE keyword is specified, VALUE is assumed.

When using FORMAT SHAPEFILE, only FILE is assumed.

**WITH clause** Use this clause to specify the rowset schema (column names and data types) of the data being queried. You can specify the columns directly (for example, WITH ( Surname CHAR(30), GivenName CHAR(30) )). You can also use the TABLE subclause to reference a table to use to obtain schema information from (for example, WITH ( TABLE dba.Employees ( Surname, GivenName ) )). You must own or have SELECT privileges on the table you specify.

When specifying columns, you can specify filler() for columns you want to skip in the input data (for example, WITH ( filler( ), Surname CHAR(30), GivenName CHAR(30) )).

**OPTION clause** Use the OPTION clause to specify parsing options to use for the input file, such as escape characters, delimiters, encoding, and so on. Supported options comprise those options for the LOAD TABLE statement that control the parsing of an input file.

- scan-option For information about each scan option, see the load options described in “LOAD TABLE statement” on page 873.
• **apply-expression**  Use this clause to specify a join condition where the right *table-expression* is evaluated for every row in the left *table-expression*. For example, you can use an apply expression to evaluate a function, procedure, or derived table for each row in a table expression.

• **contains-expression**  Use the CONTAINS clause following a table name to filter the table and return only those rows matching the full text query specified with *contains-query*. Every matching row of the table is returned together with a score column that can be referred to using *score-correlation-name*, if it is specified. If *score-correlation-name* is not specified, then the score column can be referred to by the default correlation name, contains.

With the exception of the optional correlation name argument, the CONTAINS clause takes the same arguments as those of the CONTAINS search condition.

There must be a text index on the columns listed in the CONTAINS clause.

The *contains-query* cannot be NULL or an empty string. If the text configuration settings cause all of the terms in the *contains-query* to be dropped, rows from the base table referenced by the *contains-expression* are not returned.

• **correlation-name**  Use *correlation-name* to specify a substitute name for a table or view in the FROM clause. The substitute name can then be referenced from elsewhere in the statement. For example, emp and dep are correlation names for the Employees and Departments tables, respectively:

```
SELECT Surname, GivenName, DepartmentName
FROM GROUPO.Employees emp, GROUPO.Departments dep,
WHERE emp.DepartmentID=dep.DepartmentID;
```

Correlation names can be used to distinguish between different instances of the same table. For example, the following query joins the Employee table to itself, using the Mgr correlation name, to include the surname of each employee's manager in the result:

```
SELECT Emp.EmployeeID, Emp.Surname, Dept.DepartmentName, Mgr.Surname AS ManagerName
FROM GROUPO.Employees AS Emp, GROUPO.Departments AS Dept,
GROUPO.Employees AS Mgr
WHERE Emp.DepartmentID = Dept.DepartmentID AND Emp.ManagerID = Mgr.EmployeeID;
```

• **dml-statement**  Use *dml-statement* to specify the DML statement (INSERT, DELETE, UPDATE, or MERGE) from which you want to select rows. During execution, the DML statement specified in *dml-derived-table* is executed first, and the rows affected by that DML are materialized into a temporary table whose columns are described by the REFERENCING clause. The temporary table represents the result set of *dml-derived-table*.

Use REFERENCING ( ) or REFERENCING ( NONE ) if the results do not need to be materialized into a temporary table because you are not referencing them in the query.

If you specify REFERENCING ( ) or REFERENCING ( NONE ), the updated rows are not materialized into a temporary table that represents the result set of *dml-derived-table* because they are not being referenced in the query. The temporary table in this case is an empty table. You can use this feature if you want *dml-statement* to be executed before the main statement is executed.
In the results, OLD columns contain the values as seen by the scan that finds the rows to include in the update operation. FINAL columns contain the values after referential integrity checks have been made, computed and default columns have been updated, and all triggers have fired (excluding AFTER triggers of type FOR STATEMENT).

<table>
<thead>
<tr>
<th>Statement</th>
<th>Supported table versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>FINAL</td>
</tr>
<tr>
<td>DELETE</td>
<td>OLD</td>
</tr>
<tr>
<td>UPDATE</td>
<td>FINAL and/or OLD</td>
</tr>
<tr>
<td>MERGE</td>
<td>FINAL and/or OLD</td>
</tr>
</tbody>
</table>

When specifying both OLD and FINAL names, two correlation names are used; however, these are not true correlations since they both refer to the same result set. If you specify REFERENCING (OLD AS O FINAL AS F), there is an implicit join predicate: O.rowid = F.rowid.

The INSERT statement only supports FINAL. Consequently the values of updated rows that are modified by an INSERT ON EXISTING UPDATE statement do not appear in the result set of the derived table. Instead, use the MERGE statement to perform the insert-else-update processing.

The dml-derived-table statement can only reference one updatable table; updates over multiple tables return an error. Also, selecting from dml-statement is not allowed if the DML statement appears inside a correlated subquery or common table expression because the semantics of these constructs can be unclear.

- **WITH table-hint clause** The WITH table-hint clause allows you to specify the behavior to be used only for this table, and only for this statement. Use this clause to change the behavior without changing the isolation level or setting a database or connection option. Table hints can be used for base tables, temporary tables, and materialized views.

Caution

The WITH table-hint clause is an advanced feature that should be used only if needed, and only by experienced database administrators. In addition, the setting may not be respected in all situations.

- **Isolation level related table hints** The isolation level table hints are used to specify isolation level behavior when querying tables. They specify a locking method that is used only for the specified tables, and only for the current query. You cannot specify snapshot isolation levels as table hints.

Following is the list of supported isolation level related table hints:

<table>
<thead>
<tr>
<th>Table hint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOLD-LOCK</td>
<td>Sets the behavior to be equivalent to isolation level 3. This table hint is synonymous with SERIALIZABLE.</td>
</tr>
<tr>
<td>Table hint</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NOLOCK</td>
<td>Sets the behavior to be equivalent to isolation level 0. This table hint is synonymous with READUNCOMMITTED.</td>
</tr>
<tr>
<td>READCOMMITTED</td>
<td>Sets the behavior to be equivalent to isolation level 1.</td>
</tr>
<tr>
<td>READPAST</td>
<td>Instructs the database server to ignore, instead of block on, write-locked rows. This table hint can only be used with isolation level 1. The READPAST hint is respected only when the correlation name in the FROM clause refers to a base or globally shared temporary table. In other situations (views, proxy tables, and table functions) the READPAST hint is ignored. Queries within views may utilize READPAST as long as the hint is specified for a correlation name that is a base table. The use of the READPAST table hint can lead to anomalies due to the interaction of locking and predicate evaluation within the server. In addition, you cannot use the READPAST hint against tables that are the targets of a DELETE, INSERT or UPDATE statement.</td>
</tr>
<tr>
<td>READUNCOMMITTED</td>
<td>Sets the behavior to be equivalent to isolation level 0. This table hint is synonymous with NOLOCK.</td>
</tr>
<tr>
<td>REPEATABLEREAD</td>
<td>Sets the behavior to be equivalent to isolation level 2.</td>
</tr>
<tr>
<td>SERIALIZABLE</td>
<td>Sets the behavior to be equivalent to isolation level 3. This table hint is synonymous with HOLDLOCK.</td>
</tr>
<tr>
<td>UPDLOCK</td>
<td>Indicates that rows processed by the statement from the hinted table are locked using intent locks. The affected rows remain locked until the end of the transaction. UPDLOCK works at all isolation levels and uses intent locks.</td>
</tr>
<tr>
<td>XLOCK</td>
<td>Indicates that rows processed by the statement from the hinted table are to be locked exclusively. The affected rows remain locked until the end of the transaction. XLOCK works at all isolation levels and uses write locks.</td>
</tr>
</tbody>
</table>
Note
If you are writing queries for databases that participate in MobiLink synchronization, it is recommended that you do not use the READPAST table hint in your synchronization scripts.

For more information, see:
- “download_cursor table event” [MobiLink - Server Administration]
- “download_delete_cursor table event” [MobiLink - Server Administration]
- “upload_fetch table event” [MobiLink - Server Administration]

If you are considering READPAST because your application performs many updates that affect download performance, an alternative solution is to use snapshot isolation.

○ Optimization table hint (FASTFIRSTROW) The FASTFIRSTROW table hint allows you to set the optimization goal for the query without setting the optimization_goal option to First-row. When you use FASTFIRSTROW, SQL Anywhere chooses an access plan that is intended to reduce the time to fetch the first row of the query's result.

- WITH (index-hint) clause The WITH (index-hint) clause allows you to specify index hints that override the query optimizer plan selection algorithms, and tell the optimizer exactly how to access the table using indexes. Index hints can be used for base tables, temporary tables, and materialized views.

  ○ NO INDEX Use this clause to force a sequential scan of the table (indexes are not used).
    Sequential scans may be very costly.

  ○ INDEX ([ PRIMARY KEY | FOREIGN KEY ] index-name [,...]) Use this clause to specify up to four indexes that the optimizer must use to satisfy the query.
    If any of the specified indexes cannot be used, an error is returned.

    You can specify PRIMARY KEY or FOREIGN KEY to remove ambiguity in the cases where the PRIMARY KEY index and FOREIGN KEY index on a table have the same name.

    If you specify an index name in the index hint, without the PRIMARY or FOREIGN key, and multiple indexes with the same name exist on a table, the optimizer chooses the normal index. If a normal index does not exist, the optimizer chooses the primary key index. If a primary key index does not exist, the foreign key index is used instead.

    index-name can be qualified by specifying the user ID and the table name of the index.

    The indexes specified in the INDEX clause must be indexes defined for that table; otherwise, an error is returned. For example, FROM Products WITH( INDEX (Products.xx)) returns an error if the index xx is not defined for the Products table. Likewise, FROM Products WITH( INDEX (sales_order_items.sales_order_items)) returns an error because the sales_order_items.sales_order_items index exists but is not defined for the Products table.

  ○ INDEX ONLY { ON | OFF } Use this clause to control whether an index-only retrieval of data is performed. If the INDEX (index-name...) clause is specified with INDEX ONLY ON, the
database server attempts an index-only retrieval using the specified indexes. If any of the specified indexes cannot be used in satisfying an index-only retrieval, an error is returned (for example, if there are no indexes, or if the existing indexes cannot satisfy the query).

Specify INDEX ONLY OFF to prevent an index-only retrieval.

- **FORCE INDEX (index-name)** The FORCE INDEX (index-name) syntax is provided for compatibility, and does not support specifying more than one index. This clause is equivalent to WITH (INDEX (index-name)).

- **CLUSTERED INDEX** Use this clause to specify that the optimizer must use a clustered index if one exists. The index name is not specified as only one clustered index can exist for a base table. If a clustered index doesn't exist or it cannot be used, an error is returned.

**Remarks**

Subqueries and subselects are supported as arguments to stored procedures and table functions in the FROM clause. For example, the following FROM clause is valid:

```sql
SELECT *, (SELECT 12 * ) D
FROM sa_rowgenerator( 1,( SELECT 12 x ) );
```

The SELECT, UPDATE, and DELETE statements require a table list to specify which tables are used by the statement.

**Note**

Although the FROM clause description refers to tables, it also applies to views and derived tables unless otherwise noted.

The FROM clause creates a result set consisting of all the columns from all the tables specified. Initially, all combinations of rows in the component tables are in the result set, and the number of combinations is usually reduced by JOIN conditions and/or WHERE conditions.

You cannot use an ON phrase with CROSS JOIN.

**Privileges**

The FILE clause of `openstring-expression` requires the READ FILE privilege.

The TABLE clause of `openstring-expression` requires the user to own the referenced tables, or to have the SELECT ANY TABLE privilege.

**Side effects**

None.
Standards and compatibility

**SQL/2008**  The FROM clause is a fundamental part of the SQL/2008 standard. The complexity of the FROM clause means that you should check individual components of a FROM clause against the appropriate portions of the standard. The following is a non-exhaustive list of optional SQL/2008 language features supported in SQL Anywhere:

- CROSS JOIN, FULL OUTER JOIN, and NATURAL JOIN constitute optional SQL/2008 feature F401.
- INTERSECT and INTERSECT ALL constitute optional SQL/2008 feature F302.
- EXCEPT ALL is optional language feature F304.
- derived tables are SQL/2008 language feature F591.
- procedures in the FROM clause (table functions) are feature T326. The SQL/2008 standard requires the keyword TABLE to identify the output of a procedure as a table expression, whereas in SQL Anywhere the TABLE keyword is unnecessary.
- common table expressions are optional SQL/2008 language feature T121. Using a common table expression in a derived table nested within another common table expression is language feature T122.
- recursive table expressions are feature T131. Using a recursive table expression in a derived table nested within a common table expression is optional SQL/2008 language feature T132.

The following components of the FROM clause are vendor extensions:
o KEY JOIN.

o CROSS APPLY and OUTER APPLY.

o OPENSTRING.

o a table-expression using CONTAINS (full text search).

o specifying a dml-statement as a derived table.

o all table hints, including the use of WITH, FORCE INDEX, READPAST and isolation level hints.

o LATERAL (table-expression), which is a vendor extension. LATERAL (select-statement) is in the SQL/2008 standard as optional SQL language feature T491.

Example
The following are valid FROM clauses:

```
FROM GROUPO.Employees
...
...
FROM GROUPO.Employees NATURAL JOIN GROUPO.Departments
...
...
FROM GROUPO.Customers
KEY JOIN GROUPO.SalesOrders
KEY JOIN GROUPO.SalesOrderItems
KEY JOIN GROUPO.Products
...
...
FROM GROUPO.Employees CONTAINS ('Street', ' Way ')
...
```

The following query illustrates how to use derived tables in a query:

```
SELECT Surname, GivenName, number_of_orders
FROM GROUPO.Customers JOIN
( SELECT CustomerID, COUNT(*)
  FROM GROUPO.SalesOrders
  GROUP BY CustomerID )
AS sales_order_counts( CustomerID, number_of_orders )
ON ( Customers.ID = sales_order_counts.CustomerID )
WHERE number_of_orders > 3;
```

The following query illustrates how to select rows from stored procedure result sets:

```
SELECT t.ID, t.QuantityOrdered AS q, p.name
FROM GROUPO.ShowCustomerProducts( 149 ) t JOIN GROUPO.Products p
ON t.ID = p.ID;
```

The following example illustrates how to perform a query using the OPENSTRING clause to query a file. The CREATE TABLE statement creates a table called testtable with two columns, column1 and columns2. The UNLOAD statement creates a file called testfile.dat by unloading rows from the...
RowGenerator table. The SELECT statement uses the OPENSTRING clause in a FROM clause to query testfile.dat using the schema information from both the testtable and RowGenerator tables. The query returns one row with the value 49.

```
CREATE TABLE testtable( column1 CHAR(10), column2 INT );
UNLOAD SELECT * FROM RowGenerator TO 'testfile.dat';
SELECT A.column2
FROM OPENSTRING( FILE 'testfile.dat' )
WITH ( TABLE testtable( column2 ) ) A, RowGenerator B
WHERE A.column2 = B.row_num
AND A.column2 < 50
AND B.row_num > 48;
```

The following example illustrates how to perform a query using the OPENSTRING clause to query a string value. The SELECT statement uses the OPENSTRING clause in a FROM clause to query a string value using the schema information provided in the WITH clause. The query returns two columns with three rows.

```
SELECT *
FROM OPENSTRING( VALUE '1,"First"$2,"Second"$3,"Third"' )
WITH ( c1 INT, c2 VARCHAR(30) )
OPTION ( DELIMITED BY ',' ROW DELIMITED BY '$' )
AS VALS
```

The following example illustrates how to perform a query to select the rows modified by a data manipulation statement. In this example, a warning is issued when the stock of blue items drops by more than half.

```
SELECT old_products.name, old_products.quantity, final_products.quantity
FROM
( UPDATE GROUPO.Products SET quantity = quantity - 10 WHERE color = 'Blue' )
REFERENCING ( OLD AS old_products FINAL AS final_products )
WHERE final_products.quantity < 0.5 * old_products.quantity;
```

---

**GET DATA statement [ESQL]**

Gets string or binary data for one column of the current row of a cursor.

**Syntax**

```
GET DATA  cursor-name  
COLUMN  column-num  
OFFSET  start-offset  
[  WITH TEXTPTR  ]  
{  USING DESCRIPTOR  sqlda-name  |  INTO  hostvar, ...  }
```

- `cursor-name`: identifier | hostvar
- `column-num`: integer | hostvar
- `start-offset`: integer | hostvar
- `sqlda-name`: identifier
Parameters

COLUMN clause  The value of column-num starts at one, and identifies the column whose data is to be fetched. That column must be of a string or binary type.

OFFSET clause  The start-offset indicates the number of bytes to skip over in the field value. Normally, this would be the number of bytes previously fetched. The number of bytes fetched on this GET DATA statement is determined by the length of the target host variable.

WITH TEXTPTR clause  If the WITH TEXTPTR clause is given, a text pointer is retrieved into a second host variable or into the second field in the SQLDA. This text pointer can be used with the Transact-SQL READ TEXT and WRITE TEXT statements. The text pointer is a 16-bit binary value, and can be declared as follows:

    DECL_BINARY( 16 ) textptr_var;

The WITH TEXTPTR clause can only be used with long data types (LONG BINARY, LONG VARCHAR, TEXT, IMAGE). If you attempt to use it with another data type, the error INVALID_TEXTPTR_VALUE is returned.

The total length of the data is returned in the SQLCOUNT field of the SQLCA structure.

USING DESCRIPTOR clause  The sqlda-name specifies the SQLDA (SQL Descriptor Area) that receives the fetched data. The USING DESCRIPTOR clause provides a dynamic method of specifying host variables to receive fetched data.

INTO clause  Use the INTO clause to specify the host variable that receives the fetched data. The indicator value for the target host variable is of type a_sql_len, which is currently a 16-bit value, so it is not always large enough to contain the number of bytes truncated. Instead, it contains a negative value if the field contains the NULL value, a positive value (not necessarily the number of bytes truncated) if the value is truncated, and zero if a non-NULL value is not truncated.

Similarly, if a LONG VARCHAR, LONG NVARCHAR, or LONG BINARY host variable is used with an offset greater than zero, the untrunc_len field does not accurately indicate the size before truncation.

Remarks

Get a piece of one column value from the row at the current cursor position. The cursor must be opened and positioned on a row, using FETCH.

GET DATA is usually used to fetch LONG BINARY or LONG VARCHAR fields.

Privileges

None.

Side effects

None.
See also

- “FETCH statement [ESQL] [SP]” on page 800
- “READTEXT statement [T-SQL]” on page 924
- “SET statement” on page 985
- “INTO clause” on page 823

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example uses GET DATA to fetch a binary large object (also called a BLOB).

```sql
EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY(1000) piece;
short ind;
EXEC SQL END DECLARE SECTION;
int size;
/* Open a cursor on a long varchar field */
EXEC SQL DECLARE big_cursor CURSOR FOR
SELECT long_data FROM some_table
WHERE key_id = 2;
EXEC SQL OPEN big_cursor;
EXEC SQL FETCH big_cursor INTO :piece;
for( offset = 0; ; offset += piece.len ) {
  EXEC SQL GET DATA big_cursor COLUMN 1
  OFFSET :offset INTO :piece:ind;
  /* Done if the NULL value */
  if( ind < 0 ) break;
  write_out_piece( piece );
  /* Done when the piece was not truncated */
  if( ind == 0 ) break;
}
EXEC SQL CLOSE big_cursor;
```

GET DESCRIPTOR statement [ESQL]

Retrieves information about a variable within a descriptor area, or retrieves its value.

Syntax

```
GET DESCRIPTOR descriptor-name
{ hostvar = COUNT | VALUE { integer | hostvar } assignment, ... }
```

assignment :
hostvar =
TYPE
LENGTH
PRECISION
SCALE
DATA
INDICATOR
NAME
NULLABLE
RETURNED_LENGTH
descriptor-name : identifier

Remarks

The GET DESCRIPTOR statement is used to retrieve information about a variable within a descriptor
area, or to retrieve its value.

The value \{ integer \ | \ hostvar \} specifies the variable in the descriptor area about which the information is
retrieved. Type checking is performed when doing GET...DATA to ensure that the host variable and the
descrptor variable have the same data type. LONG VARCHAR and LONG BINARY are not supported
by GET DESCRIPTOR...DATA.

If an error occurs, it is returned in the SQLCA.

Privileges

None.

Side effects

None.

See also

- “ALLOCATE DESCRIPTOR statement [ESQL]” on page 424
- “DEALLOCATE DESCRIPTOR statement [ESQL]” on page 727
- “SET DESCRIPTOR statement [ESQL]” on page 967
- “The SQL descriptor area (SQLDA)” [SQL Anywhere Server - Programming]

Standards and compatibility

- SQL/2008  GET DESCRIPTOR is part of optional SQL/2008 language feature B031, "Basic
dynamic SQL".

Example

The following example returns the type of the column with position col_num in sqlda.

```c
int get_type( SQLDA *sqlda, int col_num )
{
    EXEC SQL BEGIN DECLARE SECTION;
    int ret_type;
    int col = col_num;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL GET DESCRIPTOR sqlda VALUE :col :ret_type = TYPE;
    return( ret_type );
}
```

GET OPTION statement [ESQL]

Gets the current setting of an option. It is recommended that you use the CONNECTIONPROPERTY
function instead.
Syntax

```
GET OPTION [ userid.]option-name
{ INTO hostvar | USING DESCRIPTOR sqlda-name }
```

- `userid`: identifier, string | hostvar
- `option-name`: identifier
- `string`
- `hostvar`
- `hostvar`: indicator variable allowed
- `sqlda-name`: identifier

Remarks

The GET OPTION statement is provided for compatibility with older versions of the software. The recommended way to get the values of options is to use the CONNECTION_PROPERTY system function.

The GET OPTION statement gets the option setting of the option `option-name` for the user `userid` or for the connected user if `userid` is not specified. This is either the user's personal setting or the PUBLIC setting if there is no setting for the connected user. If the option specified is a database option and the user has a temporary setting for that option, then the temporary setting is retrieved.

If `option-name` does not exist, GET OPTION returns the warning SQLE_NOTFOUND.

Privileges

None.

Side effects

None.

See also

- "SET OPTION statement" on page 972
- "Alphabetical list of system procedures" on page 1092
- "CONNECTION_PROPERTY function [System]" on page 186

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement illustrates use of GET OPTION.

```
EXEC SQL GET OPTION 'date_format' INTO :datefmt;
```
GOTO statement [T-SQL]
Branches to a labeled statement.

Syntax
\[ label : GOTO label \]

Remarks
Any statement in a Transact-SQL procedure, trigger, or batch can be labeled. The label name is a valid identifier followed by a colon. In the GOTO statement, the colon is not used.

If you nest compound statements, you can only go to (GOTO) labels within the current compound statement and any of its ancestor compound statements. You cannot go to labels located in other compound statements that are nested within the ancestors.

Privileges
None.

Side effects
None.

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following Transact-SQL batch prints the message "yes" in the database server messages window four times:

```
DECLARE @count SMALLINT
SELECT @count = 1
restart:
    PRINT 'yes'
    SELECT @count = @count + 1
    WHILE @count <=4
    GOTO restart
```

GRANT statement
Grant roles and privileges to users and roles.

Note
The GRANT syntax related to authorities, permissions, and groups used in pre-16.0 versions of the software is still supported but deprecated. To review that syntax, see “GRANT statement (authorities and groups) (deprecated)” [SQL Anywhere Server - Database Administration].

To review the SQL Anywhere 12.0.1 GRANT statement syntax for comparison, see http://dcx.sybase.com/index.html#1201/en/dbreference/grant-statement.html.
Syntax: Grant system roles

GRANT ROLE system-role
TO grantee [, ...]

system-role :
dbo
| DIAGNOSTICS
| PUBLIC
| rs_systabgroup
| SA_DEBUG
| SYS
| SYS_REPLICATION_ADMIN_ROLE
| SYS_RUN_REPLICATION_ROLE
| SYS_SAMONITOR_ROLE
| SYS_SPATIAL_ADMIN_ROLE

Syntax: Grant user-defined roles

GRANT ROLE user-defined-role [, ...]
TO grantee [, ...]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]

Syntax: Grant compatibility roles

GRANT compatibility-role-name [, ...]
TO grantee [, ...]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
[ WITH NO SYSTEM PRIVILEGE INHERITANCE ]

compatibility-role-name :
SYS_AUTH_BACKUP_ROLE
| SYS_AUTH_DBA_ROLE
| SYS_AUTH_PROFILE_ROLE
| SYS_AUTH_READCLIENTFILE_ROLE
| SYS_AUTH_READFILE_ROLE
| SYS_AUTH_RESOURCE_ROLE
| SYS_AUTH_SA_ROLE
| SYS_AUTH_SSO_ROLE
| SYS_AUTH_VALIDATE_ROLE
| SYS_AUTH_WRITECLIENTFILE_ROLE
| SYS_AUTH_WRITEFILE_ROLE

Syntax: Grant system privileges

GRANT system-privilege [, ...]
TO grantee [, ...]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]

Syntax: Grant object-level privileges

GRANT object-level-privilege, ...
ON [ owner.]object-name
TO to-userid [, ...]
[ WITH GRANT OPTION ]
[ FROM from-userid ]

object-level-privilege :
ALL [ PRIVILEGES ]
Syntax: Grant the SET USER system privilege

```
GRANT SET USER [ ( user-list | ANY [ WITH ROLES role-list ] ) ]
TO grantee [...]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

Syntax: Grant the CHANGE PASSWORD system privilege

```
GRANT CHANGE PASSWORD [ ( user-list | ANY [ WITH ROLES role-list ] ) ]
TO grantee [...]
[ { WITH NO ADMIN | WITH ADMIN [ ONLY ] } OPTION ]
```

Parameters

- **role-name** The name of a system role, compatibility role, user-extended role, or user-defined role.

- **grantee** The user ID of a user, or the name of a role. You cannot grant privileges and roles to compatibility roles. You can grant roles and privileges to any system role; however, only the following system roles support log ins: PUBLIC, dbo, diagnostics, rs_systabgroup, and SA_DEBUG

- **WITH [NO] ADMIN OPTION clause** This clause is only applicable when granting system privileges and non-system roles. You cannot grant administrative rights on a system role; only users with the MANAGE ROLES system privilege can administer (grant and revoke) system roles.

  The default is WITH NO ADMIN OPTION, meaning that the grantee is given the roles or privileges, but not the ability to administer them.

  If WITH ADMIN OPTION is specified, each grantee is given administrative rights over each role-granted and system-privilege-granted.

  If WITH ADMIN ONLY OPTION is specified, then the grantee is given only administrative rights over the roles, but is not given the roles or privileges themselves. You can never use the WITH NO SYSTEM PRIVILEGE INHERITANCE clause with the WITH ADMIN ONLY OPTION.

  You can only use the WITH ADMIN OPTION clause with the WITH NO SYSTEM PRIVILEGE INHERITANCE clause when granting SYS_AUTH_DBA_ROLE.

- **WITH NO SYSTEM PRIVILEGE INHERITANCE** This clause prevents the grantees of a role from inheriting the role’s system privileges. Normally, when you grant a compatibility role to a user or role, the compatibility role's system privileges are available to both the role and its grantees. When you disable the inheritance of a compatibility role's system privileges, the system privileges are available only to the role, not to its grantees.

  The WITH NO SYSTEM PRIVILEGE INHERITANCE clause is provided for backwards compatibility. Disabling system privilege inheritance for a compatibility role mimics the behavior of the non-inheritable authority in version 12 and earlier databases. Enabling system privilege inheritance for a compatibility role mimics the behavior of all system roles and user-defined roles.
You can disable the inheritance of the system privileges when granting one of the following roles to users, user-extended roles, or system roles:

- SYS_AUTH_DBA_ROLE role
- SYS_AUTH_RESOURCE_ROLE
- SYS_AUTH_BACKUP_ROLE
- SYS_AUTH_VALIDATE_ROLE
- SYS_RUN_REPLICATION_ROLE

Also, you can only use the WITH ADMIN OPTION clause with the WITH NO SYSTEM PRIVILEGE INHERITANCE clause when granting SYS_AUTH_DBA_ROLE. You can never use the WITH NO SYSTEM PRIVILEGE INHERITANCE clause with the WITH ADMIN ONLY OPTION.

Disabling the system privilege inheritance for a user is only useful if you intend to convert the user to a user-extended role.

See “Changes in inheritance behavior for some authorities that became roles” [SQL Anywhere Server - Database Administration]

- **object-level-privilege**

  - **ALL privilege**  This privilege grants ALTER, DELETE, INSERT, REFERENCES, SELECT, and UPDATE privileges on tables. This privilege grants DELETE, INSERT, and UPDATE privileges on views.

  - **ALTER privilege**  This privilege allows the user to alter the named table with the ALTER TABLE statement. This privilege is not allowed for views.

  - **DELETE privilege**  This privilege allows the user to delete rows from the named table or view.

  - **INSERT privilege**  This privilege allows the user to insert rows into the named table or view.

  - **LOAD privilege**  This privilege allows the user to load the named table.

  - **REFERENCES privilege**  This privilege allows the user to create indexes on the named table and on the foreign keys that reference the named tables. If column names are specified, the user can reference only those columns. REFERENCES privileges on columns cannot be granted for views, only for tables. INDEX is a synonym for REFERENCES.

  - **SELECT privilege**  This privilege allows the user to view information in the view or table. If column names are specified, the users are allowed to view only those columns. SELECT privileges on columns cannot be granted for views, only for tables.

  - **TRUNCATE privilege**  This privilege allows the user to truncate the named object.

  - **UPDATE privilege**  This privilege allows the user to update rows in the view or table. If column names are specified, the user can update only those columns.
○ **WITH GRANT OPTION**  If WITH GRANT OPTION is specified, then the named user ID is also given permission to GRANT the same privilege to other user IDs. Members of groups do not inherit the WITH GRANT OPTION if it is granted to a group.

- **GRANT SET USER**

○ **user-list**  Specify the comma-separated list of all user IDs (targets) users that *grantee-list* can impersonate. For example: \texttt{GRANT SET USER (u1, u2, u3) ...}

○ **ANY [ WITH ROLES target_role_list ] clause**  Specify who *grantee* can impersonate without providing specific user IDs.

  If just ANY is specified, then the user can impersonate any other user. This is the default.

  If ANY WITH ROLES role-list is specified, users in *grantee-list* can impersonate anyone who has at least one of the roles listed in role-list, where role-list is a comma-separated list of roles.

○ **WITH ADMIN [ ONLY ] OPTION option**  The WITH ADMIN OPTION and WITH ADMIN ONLY OPTION clauses can only be specified with the ANY clause.

- **GRANT CHANGE PASSWORD**

○ **user-list**  Specify a comma-separated list of users that *grantee* can change passwords for.

○ **ANY [ WITH ROLES role-list ]**  Specify who *grantee* can change the password for without providing specific user IDs.

  If just ANY is specified, then the user can change any user's password. This is the default.

  If ANY WITH ROLES role-list is specified, the *grantee* can change the password for anyone who has at least one of the roles listed in role-list, where role-list is a comma-separated list of roles.

○ **WITH ADMIN [ ONLY ] OPTION option**  The WITH ADMIN OPTION and WITH ADMIN ONLY OPTION clauses can only be specified with the ANY clause.

**Remarks**

You can grant privileges on disabled objects. Privileges on disabled objects are stored in the database and become effective when the object is enabled.

With the exception of the SYS role, you can grant/revoke additional privileges and roles to/from a system role, provided you have administrative rights on the privileges and roles you are granting/revoking.

Granting SET USER to a user multiple times, specifying different user IDs they can impersonate, grants additional users to the list they can impersonate (as opposed to overwriting the previous grants).

Granting impersonation rights (GRANT SET USER) is not an indication of whether a user can successfully impersonate another user. Evaluation of whether a user can impersonate another user is done when the user ID attempts to start impersonating another user by executing a SETUSER statement. The impersonating user must have the SET USER system privilege, and must meet the at-least criteria required for impersonation.
Privileges
You must have administrative rights for each privilege and role you grant.

To grant object-level privileges, you must also have the MANAGE ANY OBJECT PRIVILEGE system
privilege, with administrative rights.

Side effects
None

Standards and compatibility
● SQL/2008 Vendor extension.

Example
Granting system privileges to roles To grant the CREATE ANY OBJECT system privilege to the
role RoleA without giving RoleA administrative rights, execute the following statement:

`GRANT CREATE ANY OBJECT TO RoleA;`

To grant RoleA the CREATE ANY OBJECT system privilege along with the ability to grant or revoke
the system privilege to and from other users and roles, execute the following statement:

`GRANT CREATE ANY OBJECT TO RoleA WITH ADMIN OPTION;`

To give RoleA administrative rights to the BACKUP DATABASE system privilege, but not the ability to
use the BACKUP DATABASE privilege, execute the following statement:

`GRANT BACKUP DATABASE TO RoleA WITH ADMIN ONLY OPTION;`

Granting roles to users To grant user Bob the role called SecurityRole without administrative rights,
execute the following statement:

`GRANT ROLE SecurityRole TO Bob;`

To grant the role RoleB all the privileges associated with RoleA but no administrative rights for RoleA,
execute the following statement:

`GRANT ROLE RoleA TO RoleB;`

To grant RoleB along with its administrative rights to the user Jane, execute the following statement:

`GRANT ROLE RoleB TO Jane WITH ADMIN OPTION;`

To grant the user John the administrative rights to RoleB, but the inability to use RoleB, execute the
following statement:

`GRANT ROLE RoleB TO John WITH ADMIN ONLY OPTION;`

Granting SET USER to users The following example specifies that User4 and User5 can
impersonate User1, User2, and User3.

`GRANT SET USER ( User1, User2, User3 ) TO User4, User5;`
The following example specifies that User1 can impersonate any user in the database. As well, User1 can grant the SET USER system privilege to other users.

```
GRANT SET USER (ANY) TO User1 WITH ADMIN OPTION;
```

The following example specifies that User1 can impersonate any user who has been granted the SYS_AUTH_BACKUP_ROLE compatibility role.

```
GRANT SET USER (ANY WITH ROLES SYS_AUTH_BACKUP_ROLE) TO User1;
```

See also
- “System privileges” [SQL Anywhere Server - Database Administration]
- “Roles” [SQL Anywhere Server - Database Administration]
- “Compatibility roles” [SQL Anywhere Server - Database Administration]
- “User-defined roles” [SQL Anywhere Server - Database Administration]
- “Impersonation” [SQL Anywhere Server - Database Administration]
- “SETUSER statement” on page 990
- “GRANT statement (authorities and groups) (deprecated)” [SQL Anywhere Server - Database Administration]

### GRANT CONNECT statement

Creates a new user, and can also be used by a user to change their own password.

**Note**

It is recommended that you use the CREATE USER statement to create users instead of the GRANT CONNECT statement.

**Syntax - Grant connect privileges**

```
GRANT CONNECT TO userid, ...
    [ AT starting-id ]
    [ IDENTIFIED BY password, ... ]
```

**Parameters**

- **CONNECT TO clause**

  Creates a new user. GRANT CONNECT can also be used by any user to change their own password. To create a user with an empty string as the password, use:

  ```
  GRANT CONNECT TO userid IDENTIFIED BY "";
  ```

  To create a user with no password, use:

  ```
  GRANT CONNECT TO userid;
  ```

  A user with no password cannot connect to the database. This is useful if you are creating a group and do not want anyone to connect to the database using the group user ID.

  The verify_password_function option can be used to specify a function to implement password rules (for example, passwords must include at least one digit). If a password verification function is used, you cannot specify more than one user ID and password in the GRANT CONNECT statement.
AT clause  This clause is for internal use by the Unload utility.

Remarks
None

Privileges
You must either be changing your own password using GRANT CONNECT, or have the MANAGE ANY USER privilege.

Side effects
Automatic commit.

See also
- “CREATE USER statement” on page 721
- “verify_password_function option” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008  Vendor extension.

Example
The following example creates a new database user named SQLTester, with password welcome.

```
GRANT CONNECT TO SQLTester
IDENTIFIED BY welcome
```

GRANT CONSOLIDATE statement [SQL Remote]
Identifies the database immediately above the current database in a SQL Remote hierarchy, that will receive messages from the current database.

Syntax
```
GRANT CONSOLIDATE
TO userid
TYPE message-system, ...
ADDRESS address-string, ...
[ SEND { EVERY | AT } hh:mm:ss ]
```

**message-system**
- FILE
- FTP
- SMTP

**address**
string

Parameters
- **userid**  The user ID for the user to be granted the privileges.
- **message-system**  One of the message systems supported by SQL Remote.
- **address**  The address for the specified message system.
Remarks
In a SQL Remote installation, the database immediately above the current database in a SQL Remote hierarchy must be granted CONSOLIDATE privilege. GRANT CONSOLIDATE is issued at a remote database to identify its consolidated database. Each database can have only one user ID with CONSOLIDATE privileges: you cannot have a database that is a remote database for more than one consolidated database.

The consolidated user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The address-name must be a valid address for the message-system, enclosed in single quotes. There can be only one consolidated user per remote database.

For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.

The GRANT CONSOLIDATE statement is required for the consolidated database to receive messages, but does not by itself subscribe the consolidated database to any data. To subscribe to data, a subscription must be created for the consolidated user ID to one of the publications in the current database. Running the database extraction utility at a consolidated database creates a remote database with the proper GRANT CONSOLIDATE statement already executed.

The optional SEND EVERY and SEND AT clauses specify a frequency at which messages are sent. The string contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If a user has been granted remote privileges without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. To run the Message Agent continuously, you must ensure that every user with REMOTE privilege has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many remote databases, the Message Agent is run periodically, and that the consolidated database will have no SEND clause specified.

Privileges
You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects
Automatic commit.

See also
- “GRANT PUBLISH statement [SQL Remote]” on page 839
- “GRANT REMOTE statement [SQL Remote]” on page 844
- “REVOKE CONSOLIDATE statement [SQL Remote]” on page 942
- “ALTER REMOTE MESSAGE TYPE statement [SQL Remote]” on page 459
- “CREATE REMOTE [MESSAGE] TYPE statement [SQL Remote]” on page 651
- “CREATE SUBSCRIPTION statement [SQL Remote]” on page 683
- “Granting CONSOLIDATE privilege” [SQL Remote]
- “SQL Remote message systems” [SQL Remote]
Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement grants consolidated status to the Sam_Singer user ID:

```sql
GRANT CONSOLIDATE TO Sam_Singer
TYPE SMTP
ADDRESS 'Singer, Samuel';
```

**GRANT CREATE statement**

Allows a user to create database objects in the specified dbspace.

**Syntax**

```
GRANT CREATE ON dbspace-name
TO userid, ...
```

**Remarks**

When you initialize a database, it contains one database file. This first database file is called the main file. All database objects and all data are placed, by default, in the main file. A dbspace is an additional database file that creates more space for data. A database can be held in up to 13 separate files (the main file and 12 dbspaces). Each table, together with its indexes, must be contained in a single database file. The SQL command CREATE DBSPACE adds a new file to the database.

**Privileges**

You must have the MANAGE ANY USER privilege.

**Side effects**

Automatic commit.

**See also**

- “Database file types” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

**GRANT EXECUTE statement**

Grants a user privilege to execute a procedure or user-defined function.

**Syntax**

```
GRANT EXECUTE ON [ owner.]{ procedure-name | user-defined-function }
TO userid, ...
```
Remarks
    None.

Privileges
    You must own the procedure, or have the MANAGE ANY OBJECT PRIVILEGE system privilege.

Side effects
    Automatic commit.

See also
    ● “System procedures” on page 1085

Standards and compatibility
    ● SQL/2008 Core feature of SQL/2008, used for granting EXECUTE privileges on stored procedures.

Example
    The following example allows user SQLTester to execute the Calculate_Report procedure.

    GRANT EXECUTE ON Calculate_Report
    TO SQLTester;

    The following SQL statement grants M_Haneef the privilege required to execute a procedure named my_procedure.

    GRANT EXECUTE
    ON my_procedure
    TO M_Haneef;

GRANT INTEGRATED LOGIN statement
    Grants a user privilege to execute a procedure or user-defined function.

Syntax
    GRANT INTEGRATED LOGIN TO user-profile-name, ...
    AS USER userid

Remarks
    The GRANT INTEGRATED LOGIN statement creates an explicit integrated login mapping between one or more Windows user or group profiles and an existing database user ID, allowing users who successfully log in to their local computer to connect to a database without having to provide a user ID or password. The user-profile-name can be of the form domain\user-name. The database user ID the integrated login is mapped to must have a password.

Privileges
    You must have the MANAGE ANY USER system privilege.
Side effects
Automatic commit.

See also
- “Windows integrated logins” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

GRANT KERBEROS LOGIN statement
Creates a Kerberos authenticated login mapping from one or more Kerberos principals to an existing database user ID.

Syntax
GRANT KERBEROS LOGIN TO client-Kerberos-principal, ...
AS USER userid

Remarks
The GRANT KERBEROS LOGIN statement creates a Kerberos authenticated login mapping from one or more Kerberos principals to an existing database user ID. This login mapping allows users who have successfully logged in to Kerberos (users who have a valid Kerberos ticket-granting ticket) to connect to a database without having to provide a user ID or password.

To use the GRANT KERBEROS LOGIN statement:

- You must already have Kerberos configured to use SQL Anywhere.
- You must already have your SQL Anywhere database configured to use Kerberos.
- The database user and the Kerberos principal must already exist.
- The database user ID the Kerberos login is mapped to must have a password.
- The client-Kerberos-principal must have the format user/instance@REALM, where instance is optional. The full principal, including the realm, must be specified, and principals that differ only in the instance or realm are treated as different.
- Principals are case sensitive and must be specified in the correct case. Mappings for multiple principals that differ only in case are not supported (for example, you cannot have mappings for both jjordan@MYREALM.COM and Jjordan@MYREALM.COM).
- If no explicit mapping is made for a Kerberos principal, and the Guest database user ID exists and has a password, then the Kerberos principal connects using the Guest database user ID (the same Guest database user ID as for integrated logins).

Privileges
You must have the MANAGE ANY USER privilege.
Side effects

Automatic commit.

See also

- “Kerberos authentication” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following SQL statement grants database login privilege to a fictitious Kerberos user pchin.

```
GRANT KERBEROS LOGIN TO "pchin@MYREALM.COM"
AS USER "kerberos-user";
```

GRANT PUBLISH statement [SQL Remote]

Grants publisher rights to a user ID. You must have the SYS_REPLICATION_ADMIN_ROLE system role to grant publisher rights.

Syntax

```
GRANT PUBLISH TO userid
```

Remarks

Each database in a SQL Remote installation is identified in outgoing messages by a user ID, called the publisher. The GRANT PUBLISH statement sets the publisher user ID associated with these outgoing messages. To change publishers, you must revoke publisher rights from the current publisher (REVOKE PUBLISH), and then grant them to the new publisher.

The database publisher can be determined by querying the special value CURRENT PUBLISHER as follows:

```
SELECT CURRENT PUBLISHER;
```

If there is no publisher, the value of CURRENT PUBLISHER is NULL.

The CURRENT PUBLISHER special value can be used as a default setting for columns. It is often useful to have a CURRENT PUBLISHER column as part of the primary key for replicating tables, as this configuration helps prevent primary key conflicts due to updates at more than one site.

To change the publisher, you must first drop the current publisher using the REVOKE PUBLISH statement, and then create a new publisher using the GRANT PUBLISH statement.

Executing this statement changes the value of the PUBLIC.db_publisher database option.

Privileges

You must have the SET ANY SYSTEM OPTION system privilege.
Side effects

Automatic commit.

See also

- “GRANT ROLE SYS_REPLICATION_ADMIN_ROLE statement [MobiLink] [SQL Remote]” on page 840
- “db_publisher option” [SQL Anywhere Server - Database Administration]
- “GRANT CONSOLIDATE statement [SQL Remote]” on page 834
- “REVOKE PUBLISH statement [SQL Remote]” on page 943
- “CREATE SUBSCRIPTION statement [SQL Remote]” on page 683
- “CURRENT PUBLISHER special value” on page 66
- “CREATE REMOTE [MESSAGE] TYPE statement [SQL Remote]” on page 651
- “GRANT REMOTE statement [SQL Remote]” on page 844
- “Creating a publisher” [SQL Remote]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement sets the database publisher to user ID JohnS.

```
GRANT PUBLISH TO JohnS;
```

The following statements revoke database publisher from JohnS and grant it to IrisM.

```
REVOKE PUBLISH FROM JohnS;
GRANT PUBLISH TO IrisM;
```

GRANT ROLE SYS_REPLICATION_ADMIN_ROLE statement [MobiLink] [SQL Remote]

Grants the SYS_REPLICATION_ADMIN_ROLE system role, which is required to perform administrative tasks related to replication, such as granting replication roles, managing publications, subscriptions, synchronization users and profiles, managing message types, and setting replication-related options.

Syntax

```
GRANT ROLE SYS_REPLICATION_ADMIN_ROLE 
TO grantee [, ... ]
[ [ WITH NO ADMIN OPTION ]
```

Parameters

- **grantee**  The user ID of a user, or the name of a role. The role can be a user-extended role, or a user-defined role. You cannot grant privileges and roles to system roles or compatibility roles.

- **WITH NO ADMIN OPTION clause**  The grantee is given the roles or privileges, but not the ability to administer them.
Remarks

SYS_REPLICATION_ADMIN_ROLE provides the following system privileges:

- MANAGE REPLICATION
- SET ANY SYSTEM OPTION
- SET ANY PUBLIC OPTION
- SET ANY USER DEFINED OPTION
- SELECT ANY TABLE
- CREATE ANY PROCEDURE
- DROP ANY PROCEDURE
- MANAGE ANY WEB SERVICE
- CREATE ANY TABLE
- DROP ANY TABLE
- SERVER OPERATOR
- MANAGE ANY USER
- MANAGE ROLES
- MANAGE ANY OBJECT PRIVILEGE

Users can also execute the following statements:

- SYNCHRONIZE PROFILE
- REMOTE RESET
- LOCK FEATURE

Privileges

You must have the MANAGE ROLES system privilege.

Side effects

Automatic commit.

See also

- “Replication-related system roles” [SQL Anywhere Server - Database Administration]
- SQL Remote: “REVOKE ROLE SYS_REPLICATION_ADMIN_ROLE statement [MobiLink] [SQL Remote]” on page 948
- SQL Remote: “SYS_RUN_REPLICATION_ROLE system role” [SQL Remote]
- SQL Remote: “Granting the SYS_REPLICATION_ADMIN_ROLE system role” [SQL Remote]
- SQL Remote: “Revoking the SYS_REPLICATION_ADMIN_ROLE system role” [SQL Remote]
- SQL Remote: “SQL Remote Message Agent (dbremote) modes” [SQL Remote]
- SQL Remote: “Security” [SQL Remote]
- MobiLink: “Synchronization initiation” [MobiLink - Client Administration]
- MobiLink: “Privileges for dbmlsync” [MobiLink - Client Administration]
- “GRANT ROLE SYS_RUN_REPLICATION_ROLE statement [MobiLink] [SQL Remote]” on page 842

Standards and compatibility

- SQL/2008 Vendor extension.
Examples

The following example grants the SYS_REPLICATION_ADMIN_ROLE role to grantee Sam_Singer:

```sql
GRANT ROLE SYS_REPLICATION_ADMIN_ROLE
TO Sam_Singer;
```

**GRANT ROLE SYS_RUN_REPLICATION_ROLE statement**

[**MobiLink**] [**SQL Remote**]

Grants the SYS_RUN_REPLICATION_ROLE system role, which is required to run the dbremote utility for replication, and the dbmlsync utility for synchronization.

**Syntax**

```sql
GRANT ROLE SYS_RUN_REPLICATION_ROLE
TO grantee
[WITH NO SYSTEM PrivILEGE INHERITANCE ]
```

**Parameters**

- **grantee**  The user ID of a user, or the name of a role. The role can be a user-extended role, or a user-defined role. You cannot grant privileges and roles to system roles or compatibility roles.

- **WITH NO SYSTEM PRIVILEGE INHERITANCE**  This clause prevents the grantees of the SYS_RUN_REPLICATION_ROLE system role from inheriting the role's system privileges. Normally, when you grant a system role to a user or role, the role's system privileges are available to both the role and its grantees. When you disable the inheritance of the SYS_RUN_REPLICATION_ROLE role's system privileges, the system privileges are available only to the role, not to its grantees.

The WITH NO SYSTEM PRIVILEGE INHERITANCE clause is provided for backwards compatibility for the SYS_RUN_REPLICATION_ROLE system role. Disabling system privilege inheritance for this role mimics the behavior of the non-inheritable authority in version 12 and earlier databases. Enabling system privilege inheritance for this role mimics the behavior of all system roles and user-defined roles.

Disabling the system privilege inheritance for a user is only useful if you intend to convert the user to a user-extended role.

See “Changes in inheritance behavior for some authorities that became roles” [SQL Anywhere Server - Database Administration]
Remarks

SYS_RUN_REPLICATION_ROLE provides the following system privileges:

- MONITOR
- BACKUP DATABASE
- DROP CONNECTION
- SELECT ANY TABLE
- SET ANY SYSTEM OPTION
- SET ANY USER DEFINED OPTION

Users can also execute the following statement:

- SYNCHRONIZE PROFILE

In MobiLink, when the connection is made from the SQL Anywhere synchronization client (dbmlsync) utility, the SYS_RUN_REPLICATION_ROLE system role enables dbmlsync to have full access to the database. Any other connection using the same user ID is granted no special authority.

In SQL Remote, when the connection is made from the Message Agent, the SYS_RUN_REPLICATION_ROLE system role enables the Message Agent to have full access to the database to make any changes contained in the messages. Any other connection using the same user ID is granted no special authority.

When a connection is made to the database in any other way, the user cannot exercise any of the privileges associated with this role. For example, the user cannot use the SELECT ANY TABLE system privilege.

The SYS_RUN_REPLICATION_ROLE system role avoids having to grant full DBA authority to a user ID, thereby avoiding security problems associated with distributing DBA user IDs and passwords.

For example, a SQL Remote user ID with the SYS_RUN_REPLICATION_ROLE system role has no extra privileges on any connection apart from the Message Agent. Even if the user ID and password for this user is widely distributed, there is no security problem. As long as the user ID has no permissions beyond CONNECT granted on the database, no one can use this user ID to access data in the database.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.
See also

- “Replication-related system roles” [SQL Anywhere Server - Database Administration]
- “REVOKE ROLE SYS_RUN_REPLICATION_ROLE statement [MobiLink] [SQL Remote]” on page 949
- MobiLink: “Synchronization initiation” [MobiLink - Client Administration]
- SQL Remote: “SYS_RUN_REPLICATION_ROLE system role” [SQL Remote]
- SQL Remote: “Granting the SYS_RUN_REPLICATION_ROLE system role” [SQL Remote]
- SQL Remote: “Revoking the SYS_RUN_REPLICATION_ROLE system role” [SQL Remote]
- SQL Remote: “SQL Remote Message Agent (dbremote) modes” [SQL Remote]
- SQL Remote: “Security” [SQL Remote]
- “Privileges for dbmlsync” [MobiLink - Client Administration]
- “GRANT ROLE SYS_REPLICATION_ADMIN_ROLE statement [MobiLink] [SQL Remote]” on page 840

Standards and compatibility

- SQL/2008        Vendor extension.

Examples

The following example grants the SYS_RUN_REPLICATION_ROLE system role to grantee Sam_Singer:

```
GRANT ROLE SYS_RUN_REPLICATION_ROLE
TO Sam_Singer;
```

**GRANT REMOTE statement [SQL Remote]**

Identifies a database immediately below the current database in a SQL Remote hierarchy, that will receive messages from the current database. These are called remote users.

**Syntax**

```
GRANT REMOTE TO userid, ...
TYPE message-system, ...
ADDRESS address-string, ...
[ SEND { EVERY | AT } send-time ]
```

**Parameters**

- **userid** The user ID for the user to be granted the privilege

- **message-system** One of the message systems supported by SQL Remote. It must be one of the following values:
  - FILE
  - FTP
  - SMTP

- **address-string** A string containing a valid address for the specified message system.

- **send-time** A string containing a time specification in the form *hh:mm:ss*. 
Remarks

In a SQL Remote installation, each database receiving messages from the current database must be granted REMOTE privilege.

The single exception is the database immediately above the current database in a SQL Remote hierarchy, which must be granted CONSOLIDATE privilege.

The remote user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The address-name must be a valid address for the message-system, enclosed in single quotes.

For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.

The GRANT REMOTE statement is required for the remote database to receive messages, but does not by itself subscribe the remote user to any data. To subscribe to data, a subscription must be created for the user ID to one of the publications in the current database, using the database extraction utility or the CREATE SUBSCRIPTION statement.

The optional SEND EVERY and SEND AT clauses specify a frequency at which messages are sent. The string contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If a user has been granted REMOTE privilege without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. To run the Message Agent continuously, you must ensure that every user with REMOTE privilege has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many consolidated databases, the Message Agent is run continuously, so that all remote databases would have a SEND clause specified. A typical setup may involve sending messages to laptop users daily (SEND AT) and to remote servers every hour or two (SEND EVERY). You should use as few different times as possible, for efficiency.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.
See also

- “REMOTE privilege” [SQL Remote]
- “Granting REMOTE privilege” [SQL Remote]
- “REVOKE REMOTE statement [SQL Remote]” on page 944
- “Subscriptions” [SQL Remote]
- “Send frequency” [SQL Remote]
- “SQL Remote message systems” [SQL Remote]
- “GRANT PUBLISH statement [SQL Remote]” on page 839
- “GRANT CONSOLIDATE statement [SQL Remote]” on page 834
- “CREATE SUBSCRIPTION statement [SQL Remote]” on page 683
- “CREATE REMOTE [MESSAGE] TYPE statement [SQL Remote]” on page 651

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement grants remote privilege to user Sam_Singer, using an SMTP email system, sending messages to the address Singer, Samuel once every two hours:

```
GRANT REMOTE TO Sam_Singer
    TYPE SMTP
    ADDRESS 'Singer, Samuel'
    SEND EVERY '02:00';
```

GRANT USAGE ON SEQUENCE statement

Grants privilege to use a specified sequence.

Syntax

```
GRANT USAGE ON SEQUENCE sequence-name
TO userid, ...
```

Remarks

None.

Privileges

You must have the MANAGE ANY OBJECT PRIVILEGE privilege.

Side effects

Automatic commit.

See also

- “Use of a sequence to generate unique values” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

GROUP BY clause

Groups columns, alias names, and functions as part of the SELECT statement.

Syntax

GROUP BY
| group-by-term, ...
| simple-group-by-term, ...  WITH ROLLUP
| simple-group-by-term, ...  WITH CUBE
| GROUPING SETS ( group-by-term, ... )

group-by-term :
simple-group-by-term
| ( simple-group-by-term, ... )
| ROLLUP ( simple-group-by-term, ... )
| CUBE ( simple-group-by-term, ... )

simple-group-by-term :
expression
| ( expression )
| ()

Parameters

GROUPING SETS clause  The GROUPING SETS clause allows you to perform aggregate operations on multiple groupings from a single query specification. Each set specified in a GROUPING SET clause is equivalent to a GROUP BY clause.

For example, the following two queries are equivalent:

```
SELECT a, b, SUM( c ) FROM t 
GROUP BY GROUPING SETS ( ( a, b ), ( a ), ( b ), ( ) );
```

```
SELECT a, b, SUM( c ) FROM t 
GROUP BY a, b 
UNION ALL 
SELECT a, NULL, SUM( c ) FROM t 
GROUP BY a 
UNION ALL 
SELECT NULL, b, SUM( c ) FROM t 
GROUP BY b 
UNION ALL 
SELECT NULL, NULL, SUM( c ) FROM t;
```

A grouping expression may be reflected in the result set as a NULL value, depending on the grouping in which the result row belongs. This may cause confusion over whether the NULL is the result of another grouping, or whether the NULL is the result of an actual NULL value in the underlying data. To distinguish between NULL values present in the input data and NULL values inserted by the grouping operator, use the GROUPING function.

Specifying an empty set of parentheses ( ) in the GROUPING SETS clause returns a single row containing the overall aggregate.

ROLLUP clause  The ROLLUP clause is similar to the GROUPING SETS clause in that it can be used to specify multiple grouping specifications within a single query specification. A ROLLUP clause of n
simple-group-by-terms generates \( n+1 \) grouping sets, formed by starting with the empty parentheses, and then appending successive group-by-terms from left to right.

For example, the following two statements are equivalent:

```sql
SELECT a, b, SUM(c) FROM t
GROUP BY ROLLUP (a, b);
```

```sql
SELECT a, b, SUM(c) FROM t
GROUP BY GROUPING SETS ((a, b), a, ());
```

You can use a ROLLUP clause within a GROUPING SETS clause.

**CUBE clause**  The CUBE clause is similar to the ROLLUP and GROUPING SETS clauses in that it can be used to specify multiple grouping specifications within a single query specification. The CUBE clause is used to represent all possible combinations that can be made from the expressions listed in the CUBE clause.

For example, the following two statements are equivalent:

```sql
SELECT a, b, SUM(c) FROM t
GROUP BY CUBE (a, b, c);
```

```sql
SELECT a, b, SUM(c) FROM t
GROUP BY GROUPING SETS ((a, b, c), (a, b), (a, c),
(b, c), a, b, c, ());
```

You can use a CUBE clause within a GROUPING SETS clause.

**WITH ROLLUP clause**  This is an alternative syntax to the ROLLUP clause, and is provided for Transact-SQL compatibility.

**WITH CUBE clause**  This is an alternate syntax to the CUBE clause, and is provided for Transact-SQL compatibility.

**Remarks**

When using the GROUP BY clause, you can group by expressions (with some limitations), columns, alias names, or functions. The result of the query contains one row for each distinct value (or set of values) of each grouping set.

The empty GROUP BY list, (), signifies the treatment of the entire input as a single group. For example, the following two statements are equivalent:

```sql
SELECT COUNT(), SUM(Salary) FROM GROUPO.Employees;
```

```sql
SELECT COUNT(), SUM(Salary) FROM GROUPO.Employees GROUP BY ();
```

**Privileges**

None.
Standards and compatibility

- **SQL/2008**  GROUP BY is a core feature of the SQL/2008 standard. GROUPING SETS, GROUP BY(), ROLLUP, and CUBE constitute portions of optional SQL/2008 language feature T431. SQL Anywhere does not support optional SQL/2008 language feature T432, "Nested and concatenated GROUPING SETS".

Vendor extensions to the GROUP BY clause include:

- WITH ROLLUP
- WITH CUBE
- the ability to specify arbitrary expressions as GROUP BY terms. In the SQL/2008 standard, every GROUP BY term must be a column reference from an underlying table in the query's FROM clause.

Examples

The following example returns a result set showing the total number of orders, and then provides subtotals for the number of orders in each year (2000 and 2001).

```
SELECT year ( OrderDate ) Year, Quarter ( OrderDate ) Quarter, count(*) Orders
FROM GROUPO.SalesOrders
GROUP BY ROLLUP ( Year, Quarter )
ORDER BY Year, Quarter;
```

Like the preceding ROLLUP operation example, the following CUBE query example returns a result set showing the total number of orders and provides subtotals for the number of orders in each year (2000 and 2001). Unlike ROLLUP, this query also gives subtotals for the number of orders in each quarter (1, 2, 3, and 4).

```
SELECT year (OrderDate) Year, Quarter (OrderDate) Quarter, count(*) Orders
FROM GROUPO.SalesOrders
GROUP BY CUBE ( Year, Quarter )
ORDER BY Year, Quarter;
```

The following example returns a result set that gives subtotals for the number of orders in the years 2000 and 2001. The GROUPING SETS operation lets you select the columns to be subtotaled instead of returning all combinations of subtotals like the CUBE operation.

```
SELECT year (OrderDate) Year, Quarter (OrderDate) Quarter, count(*) Orders
FROM GROUPO.SalesOrders
GROUP BY GROUPING SETS ( Year, Quarter )
ORDER BY Year, Quarter;
```
GROUP BY GROUPING SETS ( ( Year, Quarter ), ( Year ) )
ORDER BY Year, Quarter;

HELP statement [Interactive SQL]
Provides help in the Interactive SQL environment.

Syntax
HELP [ 'topic' ]

Remarks
The HELP statement is used to access SQL Anywhere documentation.

The topic for help can be optionally specified. You must enclose topic in single quotes. In some help formats, the topic cannot be specified; in this case, a link to a general help page for Interactive SQL appears.

You can specify the following topic values:

- SQL Anywhere error codes
- SQL statement keywords (such as INSERT, UPDATE, SELECT, CREATE DATABASE)

Privileges
None.

Side effects
None.

See also
- “Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

IF statement
Controls conditional execution of SQL statements.

Syntax
IF search-condition THEN statement-list
[ ELSEIF { search-condition | operation-type } THEN statement-list ] ...
[ ELSE statement-list ]
{ END IF | ENDIF }

850 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
Remarks

The IF statement is a control statement that allows you to conditionally execute the first list of SQL statements whose search-condition evaluates to TRUE. If no search-condition evaluates to TRUE, and an ELSE clause exists, the statement-list in the ELSE clause is executed.

Execution resumes at the first statement after the END IF.

Note
Do not confuse the syntax of the IF statement with that of the IF expression.

Privileges
None.

Side effects
None.

See also
- “BEGIN statement” on page 523
- “IF expressions” on page 23
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]
- “Search conditions” on page 40

Standards and compatibility

- SQL/2008 The IF statement is part of optional SQL/2008 language feature P002, "Computational completeness". The ENDIF keyword is a vendor extension.

Example

The following procedure illustrates the use of the IF statement:

```sql
CREATE PROCEDURE TopCustomer2 (OUT TopCompany CHAR(35), OUT TopValue INT) 
BEGIN
    DECLARE err_notfound EXCEPTION
    FOR SQLSTATE '02000';
    DECLARE curThisCust CURSOR FOR
    SELECT CompanyName, CAST(sum(SalesOrderItems.Quantity * Products.UnitPrice) AS INTEGER) VALUE
    FROM Customers
    LEFT OUTER JOIN SalesOrders
    LEFT OUTER JOIN SalesOrderItems
    LEFT OUTER JOIN Products
    GROUP BY CompanyName;
    DECLARE ThisValue INT;
    DECLARE ThisCompany CHAR(35);
    SET TopValue = 0;
    OPEN curThisCust;
    CustomerLoop:
    LOOP
        FETCH NEXT curThisCust
        INTO ThisCompany, ThisValue;
        IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop;
```
IF statement [T-SQL]

Controls conditional execution of a SQL statement, as an alternative to the Watcom SQL IF statement.

Syntax

```
IF expression statement
[ ELSE [ IF expression ] statement ]
```

Remarks

The Transact-SQL IF conditional and the ELSE conditional each control the execution of only a single SQL statement or compound statement (between the keywords BEGIN and END).

In comparison to the Watcom SQL IF statement, there is no THEN in the Transact-SQL IF statement. The Transact-SQL version also has no ELSEIF or END IF keywords.

Privileges

None.

Side effects

None.

Standards and compatibility

- SQL/2008 Transact-SQL extension.

Example

The following example illustrates the use of the Transact-SQL IF statement:

```
IF (SELECT max(ID) FROM sysobjects) < 100
RETURN
ELSE
BEGIN
  PRINT 'These are the user-created objects'
  SELECT name, type, ID
  FROM sysobjects
  WHERE ID < 100
END
```

The following two statement blocks illustrate Transact-SQL and Watcom SQL compatibility:

```
/* Transact-SQL IF statement */
IF @v1 = 0
  PRINT '0'
ELSE IF @v1 = 1
```
PRINT '1'
ELSE
  PRINT 'other'
/* Watcom SQL IF statement */
IF v1 = 0 THEN
  PRINT '0'
ELSEIF v1 = 1 THEN
  PRINT '1'
ELSE
  PRINT 'other'
END IF

**INCLUDE statement [ESQL]**

Includes a file into a source program to be scanned by the SQL preprocessor.

**Syntax**

```
INCLUDE filename
```

*filename*: SQLDA | SQLCA | string

**Remarks**

The INCLUDE statement is very much like the C preprocessor `#include` directive. The SQL preprocessor reads an embedded SQL source file and replaces all the embedded SQL statements with C-language source code. If a file contains information that the SQL preprocessor requires, include it with the embedded SQL INCLUDE statement.

Two file names are specially recognized: SQLCA and SQLDA. The following statement must appear before any embedded SQL statements in all embedded SQL source files.

```
EXEC SQL INCLUDE SQLCA;
```

This statement must appear at a position in the C program where static variable declarations are allowed. Many embedded SQL statements require variables (invisible to the application developer), which are declared by the SQL preprocessor at the position of the SQLCA include statement. The SQLDA file must be included if any SQLDAs are used.

**Privileges**

None.

**Side effects**

None.

**Standards and compatibility**

- SQL/2008   Vendor extension.
INPUT statement [Interactive SQL]

Imports data into a database table from an external file, from the keyboard, from an ODBC data source, or from a shapefile.

Syntax 1 - Import from an external file or from the keyboard

```
INPUT INTO [ owner.]table-name input-options

input-options : 
    [ ( column-name, ... ) ]
    BYTE ORDER MARK { ON | OFF } ]
    COLUMN WIDTHS ( integer, ... ]
    DELIMITED BY string
    ENCODING encoding
    ESCAPE CHARACTER character ]
    ESCAPES { ON | OFF } ]
    FORMAT input-format ]
    FROM filename | PROMPT ]
    NOSTRIP ]
    SKIP integer ]

input-format : TEXT | FIXED

encoding : identifier | string
```

Syntax 2 - Import from an ODBC data source

```
INPUT USING connection-string FROM source-table-name INTO destination-table-name 
[ CREATE TABLE { ON | OFF } ]

connection-string :
    { DRIVER=odbc-driver-name
    | DSN=odbc-data-source } [ ; { connection-parameter = value } ]
```

Syntax 3 - Loading shapefiles

```
INPUT INTO [ owner.]table-name FROM filename FORMAT SHAPEFILE 
[ SRID srid-number ]
[ ENCODING encoding ]

encoding : identifier | string
```

Parameters

- **column-name** List the names of the columns in the table in the order that corresponds to the order of the values in the input file. Use this parameter when the order of the table columns is different from the order of the values in the input file, or there are fewer columns in the input file than there are in the table.

- **BYTE ORDER MARK clause** Use this clause to specify whether to process a byte order mark (BOM) in the input file.
The BYTE ORDER MARK clause is relevant only when reading from TEXT formatted files. Attempts to use the BYTE ORDER MARK clause with FORMAT clauses other than TEXT causes an error.

The BYTE ORDER MARK clause is used only when reading or writing files encoded with UTF-8 or UTF-16 (and their variants). Attempts to use the BYTE ORDER MARK clause with any other encoding causes an error.

If the ENCODING clause is specified:

- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding with an endian such as UTF-16BE or UTF-16LE, Interactive SQL searches for a BOM at the beginning of the data. If a BOM is present, it is used to verify the endianness of the data. If you specify the wrong endian, an error is returned.
- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding without an explicit endian, Interactive SQL searches for a BOM at the beginning of the data. If a BOM is present, it is used to determine the endianness of the data. Otherwise, the operating system endianness is assumed.
- If the BYTE ORDER MARK option is ON and you specify a UTF-8 encoding, Interactive SQL searches for a BOM at the beginning of the data. If a BOM is present it is ignored.

If the ENCODING clause is not specified:

- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is ON, Interactive SQL looks for a BOM at the beginning of the input data. If a BOM is located, the source encoding is automatically selected based on the encoding of the BOM (UTF-16BE, UTF-16LE, or UTF-8) and the BOM is not considered to be part of the data to be loaded.
- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is OFF, or a BOM is not found at the beginning of the input data, encoding on the client computer is used.

**COLUMN WIDTHS clause**  
The COLUMN WIDTHS clause specifies the widths of the columns in the input file and can only be specified for FIXED format. If COLUMN WIDTHS is not specified, the widths are determined by the database column types.

**CREATE TABLE clause**  
Use the CREATE TABLE clause to specify whether to create the destination table if it does not exist. The default is ON.

**DELIMITED BY clause**  
The DELIMITED BY clause allows you to specify a string to be used as the delimiter between column values in TEXT input format. The default delimiter is a comma.

**ENCODING clause**  
The *encoding* argument specifies the encoding that is used to read the file. The ENCODING clause can only be used with the TEXT and SHAPEFILE formats.

When running Interactive SQL, the encoding that is used to import the data is determined in the following order:

- The encoding specified by the ENCODING clause (if this clause is specified).
- The encoding specified by the default_isql_encoding option (if this option is set).
- If the file has a byte-order mark, the appropriate Unicode encoding is used.
The default encoding for the computer you are running on. On English Windows computers, the default encoding is 1252 or 850.

If the input file was created using the OUTPUT statement and an encoding was specified, then the same ENCODING clause should be specified on the INPUT statement.

**ESCAPE CHARACTER clause**  The default escape character for hexadecimal codes is a backslash (\). For example, \x0A is the linefeed character.

The newline character can be specified as \n. Other characters can be specified using hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\\) is interpreted as a single backslash. A backslash followed by any character other than n, x, X, or \ is interpreted as two separate characters. For example, \q is interpreted as a backslash and the letter q.

The escape character can be changed using the ESCAPE CHARACTER clause. For example, to use the exclamation mark as the escape character, specify:

```
... ESCAPE CHARACTER '!'  
```

**ESCAPES clause**  With ESCAPES turned on (the default), characters following the escape character are interpreted as special characters by the database server. With ESCAPES turned off, the characters are read exactly as they appear in the source.

**FORMAT clause**  Use the FORMAT clause to specify the format of the input file.

If the FORMAT clause is not specified, then the input_format option sets the format. The default setting is TEXT. You can also specify the input format in Interactive SQL by clicking **Tools » Options » Import/Export** and specifying either **Text** or **Fixed** in the **Default Import Format** field.

The supported formats are:

- **TEXT**  Input lines are assumed to be characters, one row per line, with column values separated by delimiters. Alphabetic strings may be enclosed in single quotes or double quotes. Strings containing delimiters must be enclosed in either single or double quotes. If the string itself contains single or double quotes, double the quote character to use it within the string. You can use the DELIMITED BY clause to specify a different delimiter string than the default, which is a comma.

  Three other special sequences are also recognized. The two characters \n represent a newline character, \ represents a single (\), and the sequence \xDD represents the character with hexadecimal code DD.

  Omitted values are treated as NULL. If the value in that position cannot be NULL, a zero is inserted in numeric columns and an empty string in character columns.

- **FIXED**  Input lines are in fixed format. The width of the columns can be specified using the COLUMN WIDTHS clause. If they are not specified, column widths in the file must be the same as the maximum number of characters allowed by the corresponding database column.

  The FIXED format cannot be used with binary columns that contain embedded newline and end-of-file character sequences.
○ SHAPEFILE Input is in the form of an ESRI shapefile. Unlike the LOAD statement, when loading shapefiles using the INPUT statement, the shapefile must be located on the client computer. The associated .shx and .dbf files must be located in the same directory as the .shp file, and have the same base file name.

If an encoding is not specified when loading a shapefile, the default is ISO-8859-1.

Use the SRID clause to specify a SRID to associate with the geometries. If you do not specify a SRID, SRID 0 is used by default. Ideally, you should specify the same SRID as the one mentioned in the project file (.prj) for the shapefile. If that SRID is not available, use one that is equivalent. SQL Anywhere provides thousands of SRIDs you can add to the database using the sa_install_feature system procedure.

To use other formats such as, DBASE II, DBASE III, FoxPro, Lotus 123, or Excel 97, use INPUT USING.

FROM filename clause The filename can be quoted or unquoted. If the string is quoted, it is subject to the same formatting requirements as other SQL strings.

To indicate directory paths, the backslash character (\) must be represented by two backslashes when the path is quoted.

The location of a relative filename is determined as follows:

○ If the INPUT statement is executed directly in Interactive SQL, the path to filename is resolved relative to the directory in which Interactive SQL is running. For example, suppose you open Interactive SQL from the directory c:\work and execute the following statement:

```
INPUT INTO GROUPO.Employees FROM 'inputs\inputfile.dat';
```

Interactive SQL looks for c:\work\inputs\inputfile.dat.

○ If the INPUT statement resides in a .sql file, Interactive SQL first attempts to resolve the path to filename relative to the location of the file. If unsuccessful, Interactive SQL looks for filename in a path relative to the directory in which Interactive SQL is running.

For example, suppose you had a file, c:\homework\inputs.sql, that contained the following statement:

```
INPUT INTO GROUPO.Employees FROM 'inputs\inputfile.dat';
```

Interactive SQL would first look for inputfile.dat in c:\homework\inputs. If Interactive SQL does not find inputfile.dat in that location, Interactive SQL looks in the directory in which Interactive SQL is running.

FROM source-table-name clause The source-table-name parameter is a quoted string containing the name of the table in the source database. The name can be in the form database-name.owner.table-name, owner.table-name, or table-name. Use a period to separate the components, even if that is not the native separator in the source database. If the source database requires a database name, but not an owner name, the format of source-table-name must be database..table (in this case the owner name is empty). Do not quote any of the names in the parameter. For example, do not use 'dba."my-table"'; use 'dba.my-table' instead.
**INTO clause**  The name of the table to input the data into.

**PROMPT clause**  The PROMPT clause allows the user to enter values for each column in a row. When running in windowed mode, a window is displayed, allowing the user to enter the values for the new row. If you are running Interactive SQL from the command line, Interactive SQL prompts you to type the value for each column on the command line.

**NOSTRIP clause**  Normally, for TEXT input format, trailing blanks are stripped from unquoted strings before the value is inserted. NOSTRIP can be used to suppress trailing blank stripping. Trailing blanks are not stripped from quoted strings, regardless of whether the option is used. Leading blanks are stripped from unquoted strings, regardless of the NOSTRIP option setting.

**SKIP clause**  When inserting lines from a TEXT file, the SKIP clause omits the specified number of lines starting at the beginning of the file. The number specified must be a non-negative integer. The SKIP clause is for TEXT format only.

If the specified number of lines exceeds the number of lines in the file, the INPUT statement inserts no data and no error is returned.

**USING clause**  The USING clause inputs data from an ODBC data source. You can either specify the ODBC data source name with the DSN option, or the ODBC driver name and connection parameters with the DRIVER option. The `connection-parameter` parameter is a list of database-specific connection parameters.

The `odbc-data-source` parameter is the name of a user or ODBC data source name. For example, `odbc-data-source` for the SQL Anywhere sample database is SQL Anywhere 16 Demo.

The `odbc-driver-name` parameter is the ODBC driver name. For a SQL Anywhere database, `odbc-driver-name` is SQL Anywhere 16. For an UltraLite database, `odbc-driver-name` is UltraLite 16.

**Remarks**

The INPUT statement allows efficient mass insertion into a named database table. Lines of input are read either from the user via an input window (if PROMPT is specified) or from a file (if FROM filename is specified). If neither is specified, the input is read from the SQL script file that contains the INPUT statement—in Interactive SQL, this can even be directly read from the SQL Statements pane. In this case, input is ended with a line containing only the string END.

When the input is read directly from the SQL Statements pane, you must specify a semicolon before the values for the records to be inserted at the end of the INPUT statement. For example:

```sql
INPUT INTO Owner.TableName;
value1, value2, value3
value1, value2, value3
value1, value2, value3
value1, value2, value3
END
```

The END keyword (not a semicolon) terminates data for INPUT statements that do not name a file and do not include the PROMPT keyword.

If a column list is specified, the data is inserted into the specified columns of the named table. By default, the INPUT statement assumes that column values in the input file appear in the same order as they appear...
in the database table definition. If the table column order is different, you must use column-name parameter to list the table columns in the same order as the column values in the input file.

By default, the INPUT statement stops when it attempts to insert a row that causes an error. Errors can be treated in different ways by setting the on_error and conversion_error options.

Interactive SQL prints a warning on the Messages tab if any string values are truncated on INPUT. Missing values for NOT NULL columns are set to zero for numeric types and to the empty string for non-numeric types. If INPUT attempts to insert a NULL row, the input file contains an empty row.

Because the INPUT statement is an Interactive SQL command, it cannot be used in any statement executed by the server.

Privileges
You must be the owner of the table, or have the INSERT ANY TABLE system privilege, or have INSERT privilege on the table. You must also have the SELECT ANY TABLE system privilege, or have SELECT privilege on the table.

Side effects
None.

See also
- “Inserting a new row into an Interactive SQL result set” [SQL Anywhere Server - Database Administration]
- “Data import with the INPUT statement” [SQL Anywhere Server - SQL Usage]
- “Data import” [SQL Anywhere Server - SQL Usage]
- “Importing ASCII data into a new UltraLite database” [UltraLite - Database Management and Reference]
- “OUTPUT statement [Interactive SQL]” on page 907
- “INSERT statement” on page 860
- “SET OPTION statement [Interactive SQL]” on page 975
- “LOAD TABLE statement” on page 873
- “Interactive SQL” [SQL Anywhere Server - Database Administration]
- “Unload utility (dbunload)” [SQL Anywhere Server - Database Administration]
- “Support for ESRI shapefiles” [SQL Anywhere Server - Spatial Data Support]
- “Supported character sets” [SQL Anywhere Server - Database Administration]
- “Statements allowed in procedures, triggers, events, and batches” [SQL Anywhere Server - SQL Usage]
- “Tutorial: Experimenting with the spatial features” [SQL Anywhere Server - Spatial Data Support]
- “input_format option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “sa_install_feature system procedure” on page 1164
- “default_isql_encoding option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “Inserting a new row into an Interactive SQL result set” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008  Vendor extension.
Example

This example imports data into the Employees table from a fictitious TEXT file, *new_emp.inp*:

```
INPUT INTO GROUPO.Employees
    FROM new_emp.inp
    FORMAT TEXT;
```

This example copies the table `ulTest`, into a table called `saTest`. The `ulTest` table is in an UltraLite database in the file `C:\test\myULDatabase.udb`, and the `saTest` table is created in `demo.db`:

```
INPUT USING 'driver=UltraLite 16;dbf=C:\test\myULDatabase.udb'
    FROM "ulTest" INTO "saTest";
```

This example loads the shapefile `myshapefile.shp` into the `myTable` table, and assigns SRID 4269 to the geometries in the shapefile.

```
INPUT INTO myTable
    FROM 'myshapefile.shp'
    FORMAT SHAPEFILE SRID 4269
```

This example adds a new row to the `Products` table, and prompts the user to enter the values for each column.

```
INPUT INTO GROUPO.Products PROMPT;
```

This example loads data from the file `c:\temp\input.dat` into the `Employees` table. Note how the backslashes are doubled.

```
INPUT INTO GROUPO.Employees
    FROM 'c:\temp\input.dat';
```

The following example creates a table, `myInventory`, and imports data from the file `stock.txt` that contains the data but in a different column order than the table definition. To correct the order mismatch, the correct column order required for the import is specified by the *column-name* parameter after the table name in the INPUT statement. That is, *(item, Quantity)* instructs Interactive SQL to take the first column value in the input file and place it in the second column of the table, and then take the second column value in the input file and place it in the first column of the table.

```
CREATE TABLE myInventory ( 
    Quantity INTEGER, 
    item VARCHAR(60) 
); 
INPUT INTO myInventory (item, Quantity) 
    FROM stock.txt;
```

**INSERT statement**

Inserts a single row (syntax 1) or a selection of rows from elsewhere in the database (syntax 2) into a table.

**Syntax 1**

```
INSERT [ INTO ] [ owner.]table-name [ ( column-name, ... ) ] 
[ ON EXISTING ] 
[ ERROR ]
```
Parameters

VALUES clause Use the VALUES clause to specify the values to insert. To insert the default values defined for the columns, specify DEFAULT VALUES. You can also specify VALUES ( ), which is equivalent to DEFAULT VALUES. The VALUES clause also support row value constructors so that you can insert multiple rows of values in a single statement. The number and order of insert-expression values in each row-value-constructor must correspond to the column list specified in the INTO clause. If a column list is not specified, it is assumed to be the complete ordered column list for the table. If you specify an empty column list ( ), then each of the columns in the table must have a default value. If you specify multiple row value constructors, they must be separated by commas.

If an error occurs while inserting any of the rows, all of the changes are rolled back.

WITH AUTO NAME clause WITH AUTO NAME applies only to syntax 2. If you specify WITH AUTO NAME, the names of the items in the query block determine which column the data belongs in.
The query block items should be either column references or aliased expressions. Destination columns not defined in the query block are assigned their default value. This is useful when the number of columns in the destination table is very large.

The INSERT statement returns an error if the WITH AUTO NAME clause is specified and the query block contains columns that do not match columns in the target table. For example, executing the following statement returns an error indicating that the operation column in the SELECT query block cannot be found in the table:

```sql
CREATE TABLE MyTable5(
    pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
    TableName CHAR(128),
    TableNameLen INT);
INSERT INTO MyTable5 WITH AUTO NAME
SELECT length(t.table_name) AS TableNameLen,
    t.table_name AS TableName, 1 as operation
FROM SYS.SYSTAB t
WHERE table_id <= 10;
```

**ON EXISTING clause**  
The ON EXISTING clause of the INSERT statement applies to both syntaxes. It updates existing rows in a table, based on primary key lookup, with new column values. This clause can only be used on tables that have a primary key. Attempting to use this clause on tables without primary keys generates a syntax error. You cannot insert values into a proxy table with the ON EXISTING clause.

If you specify the ON EXISTING clause, the database server performs a primary key lookup for each input row. If the corresponding row does not already exist in the table, it inserts the new row. For rows that already exist in the table, you can choose to silently ignore the input row (SKIP), generate an error message for duplicate key values (ERROR), or update the old values using the values from the input row (UPDATE). By default, if you do not specify the ON EXISTING clause, attempting to insert rows into a table where the row already exists results in a duplicate key value error, and is equivalent to specifying the ON EXISTING ERROR clause. Rows that are skipped are included in the @@rowcount variable.

When using the ON EXISTING UPDATE clause, the input row is compared to the stored row. Any column values explicitly stated in the input row replace the corresponding column values in the stored row. Likewise, column values not explicitly stated in the input row result in no change to the corresponding column values in the stored row—with the exception of columns with defaults. When using the ON EXISTING UPDATE clause with columns that have defaults (including DEFAULT AUTOINCREMENT columns), you can further specify whether to update the column value with the default values by specifying ON EXISTING UPDATE DEFAULTS ON, or leave the column value as it is by specifying ON EXISTING UPDATE DEFAULTS OFF. If nothing is specified, the default behavior is ON EXISTING UPDATE DEFAULTS OFF.

**Note**  
If you anticipate many rows qualifying for the ON EXISTING condition, consider using the MERGE statement instead. The MERGE statement provides more control over the actions you can take for matching rows. It also provides a more sophisticated syntax for defining what constitutes a match.
Note

DEFAULTS ON and DEFAULTS OFF parameters do not affect values in DEFAULT TIMESTAMP, DEFAULT UTC TIMESTAMP, or DEFAULT LAST USER. For these columns, the value in the stored row is always updated during the UPDATE.

When using the ON EXISTING SKIP and ON EXISTING ERROR clauses, if the table contains default columns, the server computes the default values even for rows that already exist. As a result, default values such as AUTOINCREMENT cause side effects even for skipped rows. In this case of AUTOINCREMENT, this results in skipped values in the AUTOINCREMENT sequence. The following example illustrates this:

```
CREATE TABLE t1( c1 INT PRIMARY KEY, c2 INT DEFAULT AUTOINCREMENT );
INSERT INTO t1( c1 ) ON EXISTING SKIP VALUES( 20 );
INSERT INTO t1( c1 ) ON EXISTING SKIP VALUES( 20 );
INSERT INTO t1( c1 ) ON EXISTING SKIP VALUES( 30 );
```

The row defined in the first INSERT statement is inserted, and c2 is set to 1. The row defined in the second INSERT statement is skipped because it matches the existing row. However, the autoincrement counter still increments to 2 (but does not impact the existing row). The row defined in the third INSERT statement is inserted, and the value of c2 is set to 3. So, the values inserted for the example above are:

```
20,1
30,3
```

Caution

If you are using SQL Remote, do not replicate DEFAULT LAST USER columns. When the column is replicated the column value is set to the SQL Remote user, not the replicated value.

OPTION clause

Use this clause to specify hints for executing the statement. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION `option-value`
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- `option-name = option-value`. A `OPTION( isolation_level = ... )` specification in the query text overrides all other means of specifying isolation level for a query.

Remarks

The INSERT statement is used to add new rows to a database table.

Since text indexes and materialized views are impacted by changes to the underlying table data, consider truncating dependent text indexes or materialized views before bulk loading (LOAD TABLE, INSERT, MERGE) data into their underlying tables.

- Syntax 1

Insert a single row, or multiple rows, with the specified expression column values. Multiple rows, if specified, are delimited by additional parentheses. The keyword DEFAULT can be used to cause the default value for the column to be inserted. If the optional list of column names is given, values are inserted one for one into the specified columns. If the list of column names is not specified, the values are inserted into the table columns in the order they were created (the same order
as retrieved with SELECT *). The row is inserted into the table at an arbitrary position. (In relational databases, tables are not ordered.)

- **Syntax 2** Perform mass insertion into a table with the results of a fully general SELECT statement. Insertions are done in an arbitrary order unless the SELECT statement contains an ORDER BY clause.

If you specify column names, the columns from the SELECT list are matched ordinally with the columns specified in the column list, or sequentially in the order in which the columns were created.

Inserts can be done into views if the query specification defining the view is updatable.

Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive. So, the string 'Value' inserted into a table is always held in the database with an uppercase V and the remainder of the letters lowercase. SELECT statements return the string as Value. If the database is not case sensitive, however, all comparisons make Value the same as value, VALUE, and so on. Further, if a single-column primary key already contains an entry Value, an INSERT of value is rejected, as it would make the primary key not unique.

Inserting a significant amount of data using the INSERT statement will also update column statistics.

**Note**
To insert many rows into a table, it is more efficient to declare a cursor and insert the rows through the cursor, where possible, than to perform many separate INSERT statements. Before inserting data, you can specify the percentage of each table page that should be left free for later updates.

**Privileges**
You must be the owner of the table, or have the INSERT ANY TABLE privilege, or have INSERT privilege on the table. Additionally, if the ON EXISTING UPDATE clause is specified, you must have the UPDATE ANY TABLE system privilege, or have UPDATE privilege on the table.

**Side effects**
None.
Standards and compatibility

- **SQL/2008** The INSERT statement is a core feature of the SQL/2008 standard. The DEFAULT VALUES clause is optional SQL language feature F222, "INSERT statement: DEFAULT VALUES clause". Support for row value constructors in an INSERT statement comprises part of optional SQL language feature F641, "Row and table constructors". The VALUES keyword is a vendor extension, mandatory with SQL Anywhere to specify the list of expressions to be inserted. However, VALUES is not part of SQL/2008.

Several optional constructions are vendor extensions. These include:

- The INSERT...ON EXISTING clause is a vendor extension. A SQL/2008 compliant equivalent in many instances is the MERGE statement.
- The OPTION clause.
- The WITH AUTO NAME clause.

Examples

Add an Eastern Sales department to the database.

```sql
INSERT INTO GROUPO.Departments ( DepartmentID, DepartmentName )
VALUES ( 230, 'Eastern Sales' );
```

Create the table DepartmentHead and fill it with the names of department heads and their departments using the WITH AUTO NAME syntax.

```sql
CREATE TABLE DepartmentHead(
    pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
    DepartmentName VARCHAR(128),
    ManagerName VARCHAR(128)
);

INSERT
INTO DepartmentHead WITH AUTO NAME
SELECT GivenName || ' ' || Surname AS ManagerName,
```
DepartmentName
FROM GROUPO.Employees JOIN GROUPO.Departments
ON EmployeeID = DepartmentHeadID;

Create the table MyTable6 and populate it using the WITH AUTO NAME syntax.

CREATE TABLE MyTable6(
   pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
   TableName CHAR(128),
   TableNameLen INT );
INSERT INTO MyTable6 WITH AUTO NAME
SELECT
   length(t.table_name) AS TableNameLen,
   t.table_name AS TableName
FROM SYS.SYSTAB t
WHERE table_id <= 10;

Insert a new department, executing the statement at isolation level 3, rather than using the current isolation level setting of the database.

INSERT INTO GROUPO.Departments
   (DepartmentID, DepartmentName, DepartmentHeadID)
VALUES(600, 'Foreign Sales', 129)
OPTION( isolation_level = 3 );

The following example inserts three rows into a fictitious table, T:

INSERT INTO T (c1,c2,c3)
VALUES (1,10,100), (2,20,200), (3,30,300);

This example inserts three rows into a fictitious table, T, of four columns where each column has a default value defined:

INSERT INTO T ()
VALUES (), (), ();

**INSTALL EXTERNAL OBJECT** statement

Installs an object that can be run in an external environment.

**Syntax**

\[
\text{INSTALL EXTERNAL OBJECT} \quad \text{object-name} \\
\quad [ \text{update-mode} ] \\
\quad \text{FROM} \{ \text{FILE} \ \text{file-path} | \text{VALUE} \ \text{expression} \} \\
\quad \text{ENVIRONMENT} \ \text{environment-name} \\
\quad [ \text{AS USER} \ \text{user-name} ]
\]

environment-name :
   PERL
   | PHP

update-mode :
   NEW
   | UPDATE
Parameters

object-name  The name by which the installed object will be identified within the database.

update-mode  The update mode for the object. If the update mode is omitted, then NEW is assumed.

file-path  The location on the server computer from where the object is being installed.

environment-name  The name of the external environment in which the external object is run.

AS USER clause  Specifies the owner of the object.

Remarks

For more information about external environments, see “SQL Anywhere external environment support” [SQL Anywhere Server - Programming].

Privileges

You must have the MANAGE ANY EXTERNAL OBJECT system privilege.

Side effects

None

See also

- “SQL Anywhere external environment support” [SQL Anywhere Server - Programming]
- “ALTER EXTERNAL ENVIRONMENT statement” on page 438
- “REMOVE EXTERNAL OBJECT statement” on page 933
- “START EXTERNAL ENVIRONMENT statement” on page 997
- “STOP EXTERNAL ENVIRONMENT statement” on page 1005
- “SYSEXTERNENV system view” on page 1358
- “SYSEXTERNENVOBJECT system view” on page 1360

Standards and compatibility

- SQL/2008  Vendor extension.

Examples

In this example, you install a Perl script that is located in a file into the database.

```
INSTALL EXTERNAL OBJECT 'PerlScript'
NEW
FROM FILE 'perlfiefile.pl'
ENVIRONMENT PERL;
```

Perl code also can be built and installed from an expression, as follows:

```
INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
FROM VALUE 'sub WriteToServerConsole { print $sa_output_handle $_[0]; }'
ENVIRONMENT PERL;
```

Perl code also can be built and installed from a variable, as follows:
CREATE VARIABLE PerlVariable LONG VARCHAR;
SET PerlVariable =
    'sub WriteToServerConsole { print $sa_output_handle $_[0]; }';

INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
FROM VALUE PerlVariable
ENVIRONMENT PERL;

INSTALL JAVA statement

Makes Java classes available for use within a database.

Syntax

INSTALL JAVA
[ NEW | UPDATE ]
[ JAR jar-name ]
FROM { FILE filename | expression }
[ AS USER user-name ]

Parameters

NEW and UPDATE keyword clauses  If you specify NEW, the referenced Java class or JAR file must contain new classes, rather than updates of currently installed classes or JAR files. An error occurs if a class or JAR file with the same name exists in the database and NEW is used.

If you specify UPDATE, the referenced file may include replacements for Java classes or JAR files that are already installed in the given database.

If omitted, the default is NEW.

JAR clause  If this is specified, then the filename must designate a JAR file. JAR files typically have extensions of .jar or .zip.

Installed JAR and ZIP files can be compressed or uncompressed. Due to differences in compression schemes, it is recommended that JAR files containing textual resources be created with compression turned off.

If the JAR clause is specified, the JAR file is retained as a JAR after the classes that it contains have been installed. That JAR is the associated JAR of each of those classes. The JAR files installed in a database with the JAR clause are called the retained JAR files of the database.

The jar-name is a character string value, of up to 255 bytes long. The jar-name is used to identify the retained JAR in subsequent INSTALL JAVA UPDATE and REMOVE JAVA statements.

FROM FILE clause  Specifies the location of the Java class or JAR file to be installed.

The formats supported for filename includes fully qualified file names, such as 'c:\libs\jarname.jar' and '/usr/u/libs/jarname.jar', and relative file names, which are relative to the current working directory of the database server. If the database server's CLASSPATH includes the path to the class or JAR file, then the path does not need to be included with the file name.
The *filename* must identify either a class, JAR or ZIP file.

**FROM clause**  Expressions must evaluate to a binary type whose value contains a valid class or JAR file.

**AS USER clause**  Specifies the owner of the object.

**Remarks**

The class definition for each class is loaded by each connection's VM the first time that class is used. When you INSTALL a class, the VM on your connection is implicitly restarted. Therefore, you have immediate access to the new class. Because the VM is restarted, any values stored in Java static variables are lost, and any SQL variables with Java class types are dropped.

For other connections, the new class is loaded the next time a VM accesses the class for the first time. If the class is already loaded by a VM, that connection does not see the new class until the VM is restarted for that connection.

All installed classes can be referenced in any way by any user.

This statement is not supported on Windows Mobile.

**Privileges**

You must have the MANAGE ANY EXTERNAL OBJECT system privilege.

**See also**

- “Installing a JAR file” [SQL Anywhere Server - Programming]
- “Installing a class file” [SQL Anywhere Server - Programming]
- “REMOVE JAVA statement” on page 934
- “SYSJAR system view” on page 1367
- “SYSJARCOMPONENT system view” on page 1367
- “SYSJAVACLASS system view” on page 1368

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

This example installs a fictitious Java class named Demo, by providing the file name and location of the class.

```
INSTALL JAVA NEW
FROM FILE 'D:\JavaClass\Demo.class';
```

This example installs all the classes contained in a fictitious ZIP file, and associates them within the database with the JAR name Widgets. After the installation, the location of the ZIP file is not retained and classes must be referenced using the fully qualified class name (package name and class name).

```
INSTALL JAVA
JAR 'Widgets'
FROM FILE 'C:\Jars\Widget.zip';
```
**INTERSECT statement**

Computes the intersection between the result sets of two or more queries.

**Syntax**

```
[ WITH temporary-views ] query-block
INTERSECT [ ALL | DISTINCT ] query-block
[ ORDER BY [ integer | select-list-expression-name ] [ ASC | DESC ], ... ]
[ FOR XML xml-mode ]
[ OPTION( query-hint, ... ) ]
```

`query-block` : See “Common elements in SQL syntax” on page 421.

`query-hint` :

- `MATERIALIZED VIEW OPTIMIZATION option-value`
- `FORCE OPTIMIZATION`
- `option-name = option-value`

`option-name` : `identifier`

`option-value` :

- `hostvar` (indicator allowed)
- `string`
- `identifier`
- `number`

**Parameters**

**FOR XML clause** For information about the FOR XML clause, see “SELECT statement” on page 955.

**OPTION clause** Use this clause to specify hints for executing the statement. The following hints are supported:

- `MATERIALIZED VIEW OPTIMIZATION option-value`
- `FORCE OPTIMIZATION`
- `option-name = option-value`. A `OPTION( isolation_level = ... )` specification in the query text overrides all other means of specifying isolation level for a query.

**Remarks**

INTERSECT computes the set intersection between the result sets of two query blocks. Query blocks can be nested, and can in turn be comprised of nested SELECT statements or the set operators UNION, EXCEPT, or INTERSECT. Specifying INTERSECT alone is equivalent to specifying INTERSECT DISTINCT.

INTERSECT ALL implements bag intersection rather than set intersection. For example, if the first `query-block` contains 5 (duplicate) rows with specific values, and the second `query-block` contains 3 duplicate rows with identical values to the first, then INTERSECT ALL will return 3 rows.

The results of INTERSECT are the same as INTERSECT ALL if either `query-block` does not contain duplicate rows.
The two query-block result sets must be UNION-compatible; they must each have the same number of items in their respective SELECT lists, and the types of each expression should be comparable. If corresponding items in two SELECT lists have different data types, SQL Anywhere chooses a data type for the corresponding column in the result and automatically convert the columns in each query-block appropriately.

The column names displayed are the same column names that are displayed for the first query-block and these names are used to determine the expression names to be matched with the ORDER BY clause. An alternative way of customizing result set column names is to use a common table expression (the WITH clause).

Privileges
You must have the SELECT ANY TABLE system privilege, or be the owner of the objects specified in query-block, or have SELECT privileges on each query block.

Side effects
None.

See also
- “EXCEPT statement” on page 789
- “UNION statement” on page 1025
- “SELECT statement” on page 955
- “OPTION clause, SELECT statement” on page 962

Standards and compatibility
- SQL/2008
  INTERSECT is optional SQL language feature F302 of the SQL/2008 standard. Explicitly specifying the DISTINCT keyword with INTERSECT is optional SQL language feature T551. Specifying an ORDER BY clause with INTERSECT is SQL language feature F850. A query-block that contains an ORDER BY clause constitutes SQL/2008 feature F851. A query block that contains a row-limit clause (SELECT TOP or LIMIT) comprises optional SQL language feature F857 or F858, depending on the context. The FOR XML and OPTION clauses are vendor extensions.

- Transact-SQL
  INTERSECT is not supported by Adaptive Server Enterprise. However, both INTERSECT ALL and INTERSECT DISTINCT can be used in the Transact-SQL dialect supported by SQL Anywhere.

Example
For examples of INTERSECT usage, see “Set operators and NULL” [SQL Anywhere Server - SQL Usage].

**LEAVE statement**
Leaves a compound statement or loop.

**Syntax**

```
LEAVE statement-label
```
Remarks

The LEAVE statement is a control statement that allows you to leave a labeled compound statement or a labeled loop. Execution resumes at the first statement after the compound statement or loop.

The compound statement that is the body of a procedure or trigger has an implicit label that is the same as the name of the procedure or trigger.

Privileges

None.

Side effects

None.

See also

- “LOOP statement” on page 891
- “FOR statement” on page 805
- “BEGIN statement” on page 523
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 The LEAVE statement is part of optional SQL/2008 language feature P002, "Computational completeness".

Example

The following fragment shows how the LEAVE statement is used to leave a loop.

```
SET i = 1;
lbl: LOOP
  INSERT INTO Counters ( number )
  VALUES ( i );
  IF i >= 10 THEN
    LEAVE lbl;
  END IF;
  SET i = i + 1
END LOOP lbl
```

The following fragment uses LEAVE in a nested loop.

```
outer_loop: LOOP
  SET i = 1;
  inner_loop: LOOP
    ...
    SET i = i + 1;
    IF i >= 10 THEN
      LEAVE outer_loop
    END IF
  END LOOP inner_loop
END LOOP outer_loop
```
LOAD STATISTICS statement

Intended for internal use, this statement is used by the dbunload utility to unload column statistics from the old database into the ISYSCOLSTAT system table.

Syntax

```
LOAD STATISTICS [[ owner.]table-name.]column-name
  format-id, density, max-steps, actual-steps, step-values, frequencies
```

Parameters

- **format-id**: Internal field used to determine the format of the rest of the row in the ISYSCOLSTAT system table.
- **density**: An estimate of the weighted average selectivity of a single value for the column, not counting the selectivity of large single value selectivities stored in the row.
- **max-steps**: The maximum number of steps allowed in the histogram.
- **actual-steps**: The number of steps actually used at this time.
- **step-values**: Boundary values of the histogram steps.
- **frequencies**: Selectivities of histogram steps.

Privileges

You must have the MANAGE ANY STATISTICS system privilege.

Side effects

None.

See also

- “SYSCOLSTAT system view” on page 1351
- “Unload utility (dbunload)” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- **SQL/2008**: Vendor extension.

LOAD TABLE statement

Imports bulk data into a database table from an external file.

Syntax

```
LOAD [ INTO ] TABLE [ owner.]table-name
  [ ( column-name, ... ) ]
load-source
  [ load-option ... ]
  [ statistics-limitation-option ]
```
load-source :
{ FROM filename-expression
    | USING FILE filename-expression
    | USING CLIENT FILE client-filename-expression
    | USING VALUE value-expression
    | USING COLUMN column-expression }  
filename-expression : string | variable  
client-filename-expression : string | variable  
value-expression : expression  
column-expression :
    column-name
    FROM table-name
    ORDER BY column-list  
load-option :
    ALLOW { integer | ALL | NO } ERRORS ]
    BYTE ORDER MARK { ON | OFF }
    CHECK CONSTRAINTS { ON | OFF }
    { COMPRESSED | AUTO COMPRESSED | NOT COMPRESSED }
    { ENCRYPTED KEY 'key' | NOT ENCRYPTED }
    COMMENTS INTRODUCED BY comment-prefix
    COMPUTES { ON | OFF }
    DEFAULTS ( ON | OFF )
    DELIMITED BY string
    ENCODING encoding
    ESCAPE CHARACTER character
    ESCAPES { ON | OFF }
    FORMAT {
        TEXT
        BCP
        XML row-xpath ( column-xpath, ... ) [ NAMESPACES namespace ] }  
SHAPEFILE
    HEXADECIMAL { ON | OFF }
    MESSAGE LOG log-target
    ORDER ( ON | OFF )
    PCTFREE percent-free-space
    Q UOTE string
    QUOTES { ON | OFF }
    ROW DELIMITED BY string
    ROW LOG log-target
    SKIP integer
    STRIP { OFF | LTRIM | RTRIM | BOTH }
    WITH CHECKPOINT { ON | OFF }
    WITH { FILE NAME | ROW | CONTENT } LOGGING  
statistics-limitation-option :
    STATISTICS {
        ON [ ALL COLUMNS ]
        | ON KEY COLUMNS
        | ON ( column-list )
        | OFF }
comment-prefix : string

encoding : string

log-target : {
FILE server-filename
| CLIENT FILE client-filename
| VARIABLE variable-name
}

Parameters

column-name Use this clause to specify one or more columns to load data into. Any columns not present in the column list become NULL if DEFAULTS is OFF. If DEFAULTS is ON and the column has a default value, that value is used. If DEFAULTS is OFF and a non-nullable column is omitted from the column list, the database server attempts to convert the empty string to the column's type.

When a column list is specified, it lists the columns that are expected to exist in the file and the order in which they are expected to appear. Column names cannot be repeated. Columns whose names do not appear in the list are set to NULL/zero/empty or DEFAULT (depending on column nullability, data type, and the DEFAULTS setting). Columns that exist in the input file that are to be ignored by LOAD TABLE can be specified using filler() as a column name.

load-source Use this clause to specify the data source to load data from. There are several sources of data from which data can be loaded. The following list gives the supported load sources:

FROM clause Use this to specify a file. The filename-expression is passed to the database server as a string. The string is therefore subject to the same database formatting requirements as other SQL strings. In particular:

○ To indicate directory paths, the backslash character (\) must be represented by two backslashes.

○ The path name is relative to the database server, not to the client application.

○ You can use UNC path names to load data from files on computers other than the database server.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

USING FILE clause Use this clause to load data from a file. This is synonymous with specifying the FROM filename clause.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

When the LOAD TABLE statement is used with the USING FILE clause, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

USING CLIENT FILE clause Use this clause to load data from a file on a client computer. When the database server retrieves data from client-filename-expression, the data is not materialized in the server's
memory, so the database server limit on the size of BLOB expressions does not apply to the file. Therefore, the client file can be of an arbitrary size.

File name logging is not allowed if the table is being loaded from a client file. If the logging type is not specified, then WITH CONTENT LOGGING is used.

When the LOAD TABLE statement is used with the USING CLIENT FILE clause, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

**USING VALUE clause**  Use this clause to load data from any expression of CHAR, NCHAR, BINARY, or LONG BINARY type, or BLOB string. The following are examples of how this clause can be used:

- The following syntax uses the xp_read_file system procedure to get the values to load from the target file:

  ```sql
  ... USING VALUE xp_read_file( 'filename' )...
  ```

- The following syntax specifies the value directly, inserting two rows with values of 4 and 5, respectively:

  ```sql
  ... USING VALUE '4\n5'...
  ```

- The following syntax uses the results of the READ_CLIENT_FILE function as the value:

  ```sql
  ... USING VALUE READ_CLIENT_FILE( client-filename-expression )
  ```

  In this case, you can also specify USING CLIENT FILE client-filename-expression since they are semantically equivalent.

If the ENCODING clause is not specified in the LOAD TABLE statement, then encoding for value-expression is assumed to be in the database character set (db_charset) if value-expression is of type CHAR or BINARY, and NCHAR database character set (nchar_charset) if value-expression is of type NCHAR.

**USING COLUMN clause**  Use this clause to load data from a single column in another table. This clause is used by the database server when it replays the transaction log during recovery by replaying the LOAD TABLE...WITH CONTENT LOGGING statements. Transaction log records for LOAD TABLE...WITH CONTENT LOGGING statements comprise chunks of the original input file. When the database server encounters these chunks in the transaction log during recovery, it loads the chunks into a temporary table and then loads all the data from the original load operation.

The following clauses are required in the USING COLUMN clause:

- **table-name**  The name of the base or temporary table that contains the column to load data from. When used by the database server during recovery from the transaction log, this is the table that holds the chunks of rows to be parsed and loaded.
○ **column-name** The name of the column in *table-name* that holds the chunks of rows to be loaded.

○ **column-list** One or more columns in the destination table used to sort the rows before loading the data. *column-list* must be a verifiable unique set of values, such as a primary key or a unique index on non-nullable columns included within the column list.

**load-option clause** There are several load options you can specify to control how data is loaded. The following list gives the supported load options:

○ **ALLOW ( integer | ALL | NO ) ERRORS clause** This clause can only be specified once for the statement. The default value for this clause is 0, which means that a violation generates an error and the statement is rolled back. If you specify an integer, \( n \), then on error \( n+1 \) the database server rolls back the statement. The values ALLOW NO ERRORS and ALLOW 0 ERRORS are equivalent. This clause allows the database server to set problematic data aside and progress with the load operation.

The database server reports the last error that was encountered to the user, and this error is also logged to the MESSAGE log. Rows that are written to the ROW log can be changed and used as input to a subsequent LOAD TABLE statement.

If a ROW LOG is written to a database server or client file, its contents are written in the same character set as the original input file. If a MESSAGE LOG is written to a server or client file, its contents are written in the client's language and in the client connection's CHAR character set. If a ROW or MESSAGE LOG is written to a CHAR or NCHAR variable, it is written in the CHAR or NCHAR (respectively) character set.

○ **BYTE ORDER MARK clause** Use this clause to specify whether the server should search for and interpret a byte order mark (BOM) at the beginning of the data. By default, this option is ON. If BYTE ORDER MARK is OFF, the server does not search for a BOM.

If the ENCODING clause is specified:

- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding with an endian such as UTF-16BE or UTF-16LE, the database server searches for a BOM at the beginning of the data. If a BOM is present, it is used to verify the endianness of the data. If you specify the wrong endian, an error is returned.
- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding without an explicit endian, the database server searches for a BOM at the beginning of the data. If a BOM is present, it is used to determine the endianness of the data. Otherwise, the operating system endianness is assumed.
- If the BYTE ORDER MARK option is ON and you specify a UTF-8 encoding, the database server searches for a BOM at the beginning of the data. If a BOM is present it is ignored.

If the ENCODING clause is not specified:

- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is ON, the server looks for a BOM at the beginning of the input data. If a BOM is located, the source encoding is automatically selected based on the encoding of the BOM (UTF-16BE, UTF-16LE, or
UTF-8) and the BOM is not considered to be part of the data to be loaded. If no BOM is found, the database CHAR encoding is used.

- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is OFF, the database CHAR encoding is used, and the database server does not look for or interpret a BOM at the beginning of the data.

○ **CHECK CONSTRAINTS clause**  Use this clause to control whether constraints are checked during loading. CHECK CONSTRAINTS is ON by default, but the Unload utility (dbunload) writes out LOAD TABLE statements with CHECK CONSTRAINTS set to OFF. Setting CHECK CONSTRAINTS to OFF disables check constraints, which can be useful, for example, during database rebuilding. If a table has check constraints that call user-defined functions that are not yet created, the rebuild fails unless CHECK CONSTRAINTS is set to OFF.

○ **COMMENTS INTRODUCED BY clause**  Use this clause to specify the string used in the data file to introduce a comment. When used, LOAD TABLE ignores any line that begins with the string `comment-prefix`.

Comments are only allowed at the beginning of a new line.

If COMMENTS INTRODUCED BY is omitted, the data file must not contain comments because they are interpreted as data.

○ **COMPRESSED clause**  Specify COMPRESSED if the data being loaded is compressed in the input file. The database server decompresses the data before loading it. If you specify COMPRESSED and the data is not compressed, the LOAD fails and returns an error.

Specify AUTO COMPRESSED to allow the database server determine whether the data in the input file is compressed. If so, the database server decompresses the data before loading it.

Specify NOT COMPRESSED to indicate that the data in the input file is not compressed. You can also specify NOT COMPRESSED if the data is compressed, but you don't want the database server to decompress it. In this case, the data remains compressed in the database. However, if a file is both encrypted and compressed, you cannot use NOT ENCRYPTED without also using NOT COMPRESSED.

○ **COMPUTES clause**  By default, this option is ON, which enables recalculation of computed columns. Setting COMPUTES to OFF disables computed column recalculations. COMPUTES OFF is useful, for example, if you are rebuilding a database, and a table has a computed column that calls a user-defined function that is not yet created. The rebuild would fail unless this option was set to OFF.

The Unload utility (dbunload) writes out LOAD TABLE statements with the COMPUTES set to OFF.

○ **DEFAULTS clause**  By default, DEFAULTS is set to OFF. If DEFAULTS is OFF, any column not present in the list of columns is assigned NULL. If DEFAULTS is set to OFF and a non-nullable column is omitted from the list, the database server attempts to convert the empty string to the column's type. If DEFAULTS is set to ON and the column has a default value, that value is used.

○ **DELIMITED BY clause**  Use this clause to specify the column delimiter string. The default column delimiter string is a comma; however, it can be any string up to 255 bytes in length (for example, `DELIMITED BY '###' ...`). The delimiter you specify is a string and should be quoted. To
specify tab-delimited values, you could specify the hexadecimal escape sequence for the tab character (9)...

DELIMITED BY '\x09' ...

○ **ENCODING clause**  Use this clause to specify the character encoding used for the data being loaded into the database. The ENCODING clause cannot be used with the BCP format.

If a translation error occurs during the load operation, it is reported based on the setting of the on_charset_conversion_failure option.

Specify the BYTE ORDER clause to interpret a byte order mark in the data.

If the ENCODING clause is specified:

- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding with an endian such as UTF-16BE or UTF-16LE, the database server searches for a BOM at the beginning of the data. If a BOM is present, it is used to verify the endianness of the data. If you specify the wrong endian, an error is returned.
- If the BYTE ORDER MARK option is ON and you specify a UTF-16 encoding without an explicit endian, the database server searches for a BOM at the beginning of the data. If a BOM is present, it is used to determine the endianness of the data. Otherwise, the operating system endianness is assumed.
- If the BYTE ORDER MARK option is ON and you specify a UTF-8 encoding, the database server searches for a BOM at the beginning of the data. If a BOM is present it is ignored.

If the ENCODING clause is not specified:

- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is ON, the server looks for a BOM at the beginning of the input data. If a BOM is located, the source encoding is automatically selected based on the encoding of the BOM (UTF-16BE, UTF-16LE, or UTF-8) and the BOM is not considered to be part of the data to be loaded.
- If you do not specify an ENCODING clause and the BYTE ORDER MARK option is OFF, the database CHAR encoding is used, and the database server does not look for or interpret a BOM at the beginning of the data.

○ **ENCRYPTED clause**  Use this clause to specify encryption settings. When loading encrypted data, specify ENCRYPTED KEY followed by the key used to encrypt the data in the input file. The key can be either a string or a variable name.

Specify NOT ENCRYPTED to indicate that the data in the input file is not encrypted. You can also specify NOT ENCRYPTED if the data is encrypted, but you don't want the database server to decrypt it. In this case, the data remains encrypted in the database. However, if a file is both encrypted and compressed, you cannot use NOT ENCRYPTED without also using NOT COMPRESSED.

○ **ESCAPE CHARACTER clause**  Use this clause to specify the escape character used in the data. The default escape character for characters stored as hexadecimal codes and symbols is a backslash (\), so \x0A is the linefeed character, for example. This can be changed using the ESCAPE CHARACTER clause. For example, to use the exclamation mark as the escape character, you would enter:

```
ESCAPE CHARACTER '!'```
It is recommended that the string you specify for the escape character is no longer than one multibyte character.

- **ESCAPES clause**  Use this clause to control whether to recognize escape characters. With ESCAPES turned ON (the default), characters following the escape character (which defaults to \) are recognized and interpreted as special characters by the database server. Newline characters can be included as the combination \n, and other characters can be included in data as hexadecimal ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters ( \\ ) is interpreted as a single backslash. A backslash followed by any character other than n, x, X, or \ is interpreted as two separate characters. For example, \q inserts a backslash and the letter q. It is recommended that the string you specify for the escape character is no longer than one multibyte character.

- **FORMAT TEXT**  If you choose FORMAT TEXT (the default), input lines are assumed to be characters (as defined by the ENCODING option), one row per line, with values separated by the column delimiter string.

- **FORMAT BCP**  Specify FORMAT BCP to load Adaptive Server Enterprise-generated BCP out files.

- **FORMAT SHAPEFILE**  Specify FORMAT SHAPEFILE to load ESRI shapefiles. The shapefile must be on the database server computer and must be loaded using FROM filename-expression or USING FILE filename-expression, where filename-expression refers to an ESRI shapefile with the .shp file extension. The associated .shx and .dbf files must be located in the same directory as the .shp file, and have the same base file name.

For FORMAT SHAPEFILE, the encoding defaults to ISO-8859-1 if the ENCODING clause is not specified.

If you specify FORMAT SHAPEFILE, the following load options are allowed:

- CHECK CONSTRAINTS
- COMPUTES
- DEFAULTS
- ENCODING
- ORDER
- PCTFREE
- WITH CHECKPOINT
- WITH .... LOGGING

The LOAD TABLE statement gets the SRID from the second column type that you are loading into. For example, if you created the second column with type ST_Geometry(SRID=4326), then the geometries are loaded using SRID 4326. If your second column has type ST_Geometry (no explicit SRID), then the geometries are loaded using SRID 0.
If you specify FORMAT XML, the following load options are allowed:

- BYTE ORDER MARK
- CHECK CONSTRAINTS
- COMPRESSED
- COMPUTES
- DEFAULTS
- ENCODING
- ENCRYPTED
- ORDER
- PCTFREE
- WITH CHECKPOINT
- WITH...LOGGING

If you use FORMAT XML, the input file is parsed in the same way as a query that uses the OPENXML operator. The arguments of the SQL statement correspond to the system procedure parameters as follows:

<table>
<thead>
<tr>
<th>LOAD TABLE statement clause</th>
<th>OPENXML operator argument</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>row-xpath</td>
<td>xpath</td>
<td></td>
</tr>
<tr>
<td></td>
<td>flags</td>
<td>There is no way to specify a value with FORMAT XML that corresponds to the flags argument of OPENXML.</td>
</tr>
<tr>
<td>NAMESPACES</td>
<td>namespaces</td>
<td></td>
</tr>
</tbody>
</table>

The FORMAT XML clause uses the following parameters:

- **row-xpath** A string or variable containing an XPath query. XPath allows you to specify patterns that describe the structure of the XML document you are querying. The XPath pattern included in this argument selects the nodes from the XML document. Each node that matches the XPath query in the row-xpath argument generates one row in the table.

Metaproperties can only be specified in FORMAT XML clause row-xpath arguments. A metaproperty is accessed within an XPath query as if it was an attribute. If namespaces is not specified, then by default the prefix mp is bound to the Uniform Resource Identifier (URI) urn:ianywhere-com:sa-xpath-metaprop. If namespace is specified, this URI must be bound to mp or some other prefix to access metaproperties in the query. Metaproperty names are case sensitive. The following metaproperties are supported:

- **@mp:id** returns an ID for a node that is unique within the XML document. The ID for a given node in a given document may change if the database server is restarted. The value of this metaproperty increases with document order.

- **@mp:localname** returns the local part of the node name, or NULL if the node does not have a name.
- **@mp:prefix** returns the prefix part of the node name, or NULL if the node does not have a name or if the name is not prefixed.

- **@mp:namespaceuri** returns the URI of the namespace that the node belongs to, or NULL if the node is not in a namespace.

- **@mp:xmltext** returns a subtree of the XML document in XML form. For example, when you match an internal node, you can use this metaproperty to return an XML string, rather than the concatenated values of the descendant text nodes.

- **column-xpath** A string or variable that specifies the schema of the result set and how the value is found for each column in the result set. If a FORMAT XML clause expression matches more than one node, then only the first node in the document order is used. If the node is not a text node, then the result is found by appending all the text node descendants. If a FORMAT XML clause expression does not match any nodes, then the column for that row is NULL.

- **namespace** A string or variable containing an XML document. The in-scope namespaces for the query are taken from the root element of the document.

- **HEXADECIMAL clause** Use this clause to specify whether to read binary values as hexadecimal values. By default, HEXADECIMAL is ON. With HEXADECIMAL ON, binary column values are read as 0xnnnnnn..., where 0x is a zero followed by an x, and each n is a hexadecimal digit. It is important to use HEXADECIMAL ON when dealing with multibyte character sets.

The HEXADECIMAL clause can be used only with the FORMAT TEXT clause.

- **MESSAGE LOG clause** This clause can only be specified once for the statement. When an error is encountered while inserting or parsing a row, the database server writes the error to the specified location.

- **ORDER clause** Use this clause to specify whether to sort the data when loading. The default for ORDER is ON. If ORDER is ON, and a clustered index has been declared, then LOAD TABLE sorts the input data according to the clustered index and inserts rows in the same order. If the data you are loading is already sorted, you should set ORDER to OFF.

- **PCTFREE clause** Use this clause to specify the percentage of free space you want to reserve for each table page. This setting overrides any permanent setting for the table, but only for the duration of the load, and only for the data being loaded. The value percent-free-space is an integer between 0 and 100. A value of 0 specifies that no free space is to be left on each page—each page is to be fully packed. A high value causes each row to be inserted into a page by itself.

- **QUOTE clause** The QUOTE clause is for TEXT data only; the string is placed around string values. The default is a single quote (apostrophe).

- **QUOTES clause** Use this clause to specify whether strings are enclosed in quotes. When QUOTES is set to ON (the default), the LOAD TABLE statement expects strings to be enclosed in quote characters. If the QUOTES clause is omitted, the quote character is either an apostrophe (single quote) or a quotation mark (double quote) and the first such character encountered in a string is treated as the quote character for the string. Strings must be terminated by a matching quote.
When QUOTES is set to ON, column delimiter strings can be included in column values. Also, quote characters are assumed not to be part of the value. Therefore, the following line is treated as two values, not three, despite the presence of the comma in the address. Also, the quotes surrounding the address are not inserted into the database.

'123 High Street, Anytown',(715)398-2354

To include a quote character in a value, when QUOTES is set to ON, you must use two quotes. The following line includes a value in the third column that is a single quote character:

'123 High Street, Anytown',(715)398-2354','''

○ **ROW DELIMITED BY clause**  Use this clause to specify the string that indicates the end of an input record. The default delimiter string is a newline (\n); however, it can be any string up to 255 bytes in length (for example, ROW DELIMITED BY '###'). If you wanted to specify tab-delimited values, you could specify the hexadecimal escape sequence for the tab character (9), ROW DELIMITED BY '\x09'. If your delimiter string contains a \n, it matches either \n or \n.

○ **ROW LOG clause**  This clause can only be specified once for the statement. When an error is encountered while inserting or parsing a row, the database server writes an image of the input row to the specified location in addition to reporting the row to the user.

○ **SKIP clause**  Use this clause to specify whether to ignore lines at the beginning of a file. The integer argument specifies the number of lines to skip. You can use this clause to skip over a line containing column headings, for example. If the row delimiter is not the default (newline), then skipping may not work correctly if the data contains the row delimiter embedded within a quoted string.

○ **STRIP clause**  Use this clause to specify whether unquoted values should have leading or trailing blanks stripped off before they are inserted. The STRIP option accepts the following options:

  - **STRIP OFF**  Do not strip off leading or trailing blanks.
  - **STRIP LTRIM**  Strip leading blanks.
  - **STRIP RTRIM**  Strip trailing blanks.
  - **STRIP BOTH**  Strip both leading and trailing blanks.

The STRIP behavior is tied to the QUOTES clause. If you specify QUOTES OFF, then STRIP OFF, STRIP LTRIM, STRIP RTRIM, and STRIP BOTH, work exactly as their wording suggests. If you do not specify a QUOTES clause, or you specify QUOTES ON, unquoted strings are always left-trimmed and right-trimmed (however, you can specify STRIP OFF or STRIP LTRIM if you don't want the strings to be right-trimmed as well).

○ **WITH CHECKPOINT clause**  Use this clause to specify whether to perform a checkpoint. The default setting is OFF. If this clause is set to ON, a checkpoint is issued after successfully completing and logging the statement. If this clause is set to ON, and the database requires automatic recovery before a checkpoint is issued, the data file used to load the table must be present for the recovery to complete successfully if you use FILE NAME LOGGING. If WITH CHECKPOINT ON is specified,
and recovery is subsequently required, recovery begins after the checkpoint, and the data file does not need to be present.

The data files are required, regardless of what is specified for this clause, if the database becomes corrupt and you need to use a backup and apply the current log file if you use FILE NAME LOGGING.

○ **WITH { FILE NAME | ROW | CONTENT } LOGGING**  Use this clause to control the level of detail logged in the transaction log during a load operation. The levels of logging are as follows:

- **WITH FILE NAME LOGGING clause**  The WITH FILE NAME LOGGING clause causes only the LOAD TABLE statement to be recorded in the transaction log. To guarantee consistent results when the transaction log is used during recovery, the file used for the original load operation must be present in its original location, and must contain the original data. This level of logging has the best performance; however, you should not use it if your database is involved in mirroring or synchronization. Also, this level cannot be used when loading from an expression or a client file.

  When you do not specify a logging level in the LOAD TABLE statement, WITH ROW LOGGING is the default level when specifying:

  ○ FROM filename-expression
  ○ USING FILE filename-expression

- **WITH ROW LOGGING clause**  The WITH ROW LOGGING clause causes each row that is loaded to be recorded in the transaction log as an INSERT statement. This level of logging is recommended for databases involved in synchronization and is the default for database mirroring when using FROM filename-expression or USING FILE filename-expression. However, when loading large amounts of data, this logging type can affect performance, and results in a much longer transaction log.

  If there are no non-deterministic values, WITH CONTENT LOGGING likely results in better performance

  This level is also ideal for databases where the table being loaded into contains non-deterministic values, such as computed columns, or CURRENT TIMESTAMP defaults.

- **WITH CONTENT LOGGING clause**  The WITH CONTENT LOGGING clause causes the database server to copy the input file to the transaction log in chunks. These chunks can be reconstituted into a copy of the input file later, for example during recovery from the transaction log. When loading large amounts of data, this logging type has a very low impact on performance, and offers increased data protection, but it does result in a longer transaction log. This level of logging is recommended for databases involved in mirroring, or where it is desirable to not maintain the original data files for later recovery provided there are no non-deterministic values.

  The WITH CONTENT LOGGING clause cannot be used if the database is involved in synchronization. The WITH CONTENT LOGGING clause is required if the table is being loaded from a client file.

  When you do not specify a logging level in the LOAD TABLE statement, WITH CONTENT LOGGING is the default level when specifying:
Using a LOAD TABLE statement on a primary or root server, WITH CONTENT LOGGING is the default.

**statistics-limitation-option**

Allows you to limit the columns for which statistics are generated during the execution of LOAD TABLE. Otherwise, statistics are generated for all columns. You should only use this clause if you are certain that statistics will not be used on some columns. You can specify ON ALL COLUMNS (the default), OFF, ON KEY COLUMNS, or a list of columns for which statistics should be generated.

**Remarks**

LOAD TABLE allows efficient mass insertion into a database table from a file. LOAD TABLE is more efficient than the Interactive SQL statement INPUT.

LOAD TABLE uses an exclusive schema lock. For base tables, global temporary tables, and local temporary tables, a commit is performed.

If you attempt to use LOAD TABLE on a table on which an immediate text index is built, or that is referenced by an immediate view, the load fails. This does not occur for non-immediate text indexes or materialized views; however, it is strongly recommended that you truncate the data in dependent indexes and materialized views before executing the LOAD TABLE statement, and then refresh the indexes and views after.

Do not use the LOAD TABLE statement on a temporary table for which ON COMMIT DELETE ROWS was specified, either explicitly or by default, at creation time. However, you can use LOAD TABLE if ON COMMIT PRESERVE ROWS or NOT TRANSACTIONAL was specified.

With FORMAT TEXT, a NULL value is indicated by specifying no value. For example, if three values are expected and the file contains 1, 'Fred', then the values inserted are 1, NULL, and Fred. If the file contains 1, 2, then the values 1, 2, and NULL are inserted. Values that consist only of spaces are also considered NULL values. For example, if the file contains 1, 'Fred', then values 1, NULL, and Fred are inserted. All other values are considered not NULL. For example, '' (a single quote followed by single quote) is an empty string. 'NULL' is a string containing four letters.

If a column being loaded by LOAD TABLE does not allow NULL values and the file value is NULL, then numeric columns are given the value 0 (zero), character columns are given an empty string (''). If a column being loaded by LOAD TABLE allows NULL values and the file value is NULL, then the column value is NULL (for all types).

If the table contains columns a, b, and c, and the input data contains a, b, and c, but the LOAD statement specifies only a and b as columns to load data into, the following values are inserted into column c:

- if DEFAULTS ON is specified, and column c has a default value, the default value is used.
• if column c does not have a default defined for it and it allows NULLs, then a NULL is used.

• if column c does not have a default defined for it and it does not allow NULLs, then either a zero (0) or an empty string ("") is used, or an error is returned, depending on the data type of the column.

• **LOAD TABLE and database mirroring**

• **LOAD TABLE and column statistics** To create histograms on table columns, LOAD TABLE captures column statistics when it loads data. The histograms are used by the optimizer.

Following are additional tips about loading and column statistics:

○ LOAD TABLE saves statistics on base tables for future use. It does not save statistics on global temporary tables.

○ If you are loading into an empty table that may have previously contained data, it may be beneficial to drop the statistics for the column before executing the LOAD TABLE statement.

○ If column statistics exist when LOAD TABLE is performed on a column, statistics for the column are not recalculated. Instead, statistics for the new data are inserted into the existing statistics. If the existing column statistics are out-of-date, then they are still out of date after loading new data into the column. If you suspect that the column statistics are out of date, then you should consider updating them either before, or after, executing the LOAD TABLE statement.

○ LOAD TABLE adds statistics only if the table has five or more rows. If the table has at least five rows, histograms are modified as follows:

<table>
<thead>
<tr>
<th>Data already in table?</th>
<th>Histogram present?</th>
<th>Action taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Integrate changes into the existing histograms</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Do not build histograms</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Integrate changes into the existing histograms</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Build new histograms</td>
</tr>
</tbody>
</table>

○ LOAD TABLE does not generate statistics for columns that contain NULL values for more than 90% of the rows being loaded.

• **Using dynamically constructed file names** You can execute a LOAD TABLE statement with a dynamically constructed file name by assigning the file name to a variable and using the variable name in the LOAD TABLE statement.

The LOAD TABLE statement requires an exclusive schema lock.

**Privileges**

The required privilege depend on the -gl server option.
If the -gl option is set to ALL, one of the following must be true:

- you are the owner of the table
- you have LOAD privilege on the table
- you have the LOAD ANY TABLE system privilege
- you have the ALTER ANY TABLE system privilege

If the -gl option is set to DBA, you must have the LOAD ANY TABLE or ALTER ANY TABLE system privilege.

If the -gl option is set to NONE, LOAD TABLE is not permitted.

When loading from a file on a client computer:

- READ CLIENT FILE privilege is also required.
- Read privileges are required on the directory being read from.
- The allow_read_client_file database option must be enabled.
- The read_client_file secure feature must be enabled.

**Side effects**

Automatic commit.

Inserts are not recorded in the transaction log file unless WITH ROW LOGGING clause is specified, so the inserted rows may not be recovered in the event of a failure depending upon the logging type. The original file is required if you need to recover the rows and WITH FILE NAME LOGGING is used. In addition, the LOAD TABLE statement should not be used without the WITH ROW LOGGING clause in databases that are used as MobiLink clients, or in a database involved in SQL Remote replication, because these technologies replicate changes through analysis of the log file.

The LOAD TABLE statement does not fire any triggers associated with the table.

A checkpoint is carried out at the beginning of the operation. A second checkpoint is performed at the end if WITH CHECKPOINT ON is specified.

Column statistics are updated if a significant amount of data is loaded.
See also

- “Data import and export” [SQL Anywhere Server - SQL Usage]
- “Data import with the LOAD TABLE statement” [SQL Anywhere Server - SQL Usage]
- “Supported character sets” [SQL Anywhere Server - Database Administration]
- “UNLOAD statement” on page 1027
- “progress_messages option” [SQL Anywhere Server - Database Administration]
- “-sf database server option” [SQL Anywhere Server - Database Administration]
- “allow_read_client_file option” [SQL Anywhere Server - Database Administration]
- “-gl database server option” [SQL Anywhere Server - Database Administration]
- “on_charset_conversion_failure option” [SQL Anywhere Server - Database Administration]
- “Support for ESRI shapefiles” [SQL Anywhere Server - Spatial Data Support]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

Suppose you create a table, myTable, as follows:

```sql
CREATE TABLE myTable( a CHAR(100), let_me_default INT DEFAULT 1, c CHAR(100) );
```

Then you create an input file called `c:\temp\input.txt` and put the following data in it:

```sql
ignore_me, this_is_for_column_c, this_is_for_column_a
```

Now, you load the data from `c:\temp\input.txt` into myTable as follows:

```sql
LOAD TABLE myTable ( filler(), c, a ) FROM 'c:\temp\input.txt' FORMAT TEXT DEFAULTS ON;
```

The command `SELECT * FROM myTable` yields the result set:

<table>
<thead>
<tr>
<th>a</th>
<th>let_me_default</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>this_is_for_column_a</td>
<td>1</td>
<td>this_is_for_column_c</td>
</tr>
</tbody>
</table>

The following example executes the LOAD TABLE statement with a dynamically constructed file name, via the EXECUTE IMMEDIATE statement:

```sql
CREATE PROCEDURE LoadData( IN from_file LONG VARCHAR )
BEGIN
    DECLARE path LONG VARCHAR;
    SET path = 'd:\data\' || from_file;
    LOAD TABLE MyTable FROM path;
END;
```

The following example loads UTF-8-encoded table data from a fictitious file into mytable:

```sql
LOAD TABLE mytable FROM 'c:\temp\mytable_data_in_utf8.dat' ENCODING 'UTF-8';
```

In this example, lines in a fictitious file `c:\temp\input2.dat` that start with // are ignored.
LOCK FEATURE statement
Prevents other concurrent connections from using a database server feature.

Syntax
LOCK FEATURE feature-name { ON | OFF }

Parameters
feature-name: The name of the feature to be locked or unlocked. Specify all to unlock all the features locked by a connection.

ON | OFF Specify ON to prevent other connections from using the feature. Specify OFF to allow connections to use the feature.

Remarks
You cannot lock a feature more than once for the same connection. If you attempt to unlock a feature that is not locked by the current connection and you do not specify all as the feature name, an error is returned. When a feature is locked by two or more connections, the feature must be unlocked by all connections before it can be used by other connections. Feature locks created by a connection are removed when the connection is dropped. Feature locks are removed when the database server is shut down.

When the synchronization schema feature is locked, the following statements cannot be executed by other connections:

● START SYNCHRONIZATION SCHEMA CHANGE
● CREATE SYNCHRONIZATION SUBSCRIPTION
● DROP SYNCHRONIZATION SUBSCRIPTION
● ALTER SYNCHRONIZATION SUBSCRIPTION
● ALTER PUBLICATION

Privileges
You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects
None

See also
● “LOCK TABLE statement” on page 890
Standards and compatibility

- SQL/2008  Vendor extension.

Example
The following statement prevents other connections from using the synchronization schema feature:

    LOCK FEATURE 'synchronization schema' ON;

LOCK TABLE statement

Prevents other concurrent transactions from accessing or modifying a table.

Syntax

    LOCK TABLE  table-name
    [ WITH HOLD ]
    IN ( SHARE | EXCLUSIVE ) MODE

Parameters

- **table-name**  The name of the table. The table must be a base table, not a view. As temporary table data is local to the current connection, locking global or local temporary tables has no effect.

- **WITH HOLD clause**  Specify this clause to lock the table until the end of the connection. If the clause is not specified, the lock is released when the current transaction is committed or rolled back.

- **IN SHARE MODE clause**  Specify this clause to obtain a shared table lock on the table, preventing other transactions from modifying the table but allowing them read access. If a transaction puts a shared lock on a table, it can change data in the table provided no other transaction holds a lock of any kind on the row(s) being modified. Read locks on individual rows are not acquired when the IN SHARE MODE clause is selected.

- **IN EXCLUSIVE MODE clause**  Specify this clause to obtain an exclusive table lock on the table, preventing other transactions from accessing the table. No other transaction can execute queries, updates, or any other action against the table. If a table is locked exclusively with a statement such as LOCK TABLE...IN EXCLUSIVE MODE, the default behavior is to not acquire row locks for the table. This behavior can be disabled by setting the subsume_row_locks option to Off.

Remarks

The LOCK TABLE statement allows direct control over concurrency at a table level, independent of the current isolation level.

While the isolation level of a transaction generally governs the kinds of locks that are set when the current transaction executes a request, the LOCK TABLE statement allows more explicit control locking of the rows in a table.

You cannot execute the LOCK TABLE statement on a view. However, if you execute the LOCK TABLE statement on a base table, a shared schema lock is created, which locks dependent views. A shared schema lock persists until the transaction is committed or rolled back.
Privileges
To lock a table in SHARE mode, you must be the owner of the table, or have SELECT privilege on the table, or have the SELECT ANY TABLE system privilege.

To lock a table in EXCLUSIVE mode, one of the following must be true:

- you must be the owner
- you must have DELETE, UPDATE, INSERT, ALTER, LOAD, and TRUNCATE privilege on the table
- you must have the DELETE ANY TABLE, UPDATE ANY TABLE, INSERT ANY TABLE, ALTER ANY TABLE, or LOAD ANY TABLE, and TRUNCATE ANY TABLE system privilege.

Side effects
Other transactions that require access to the locked table may be delayed or blocked.

See also
- “Table locks” [SQL Anywhere Server - SQL Usage]
- “SELECT statement” on page 955
- “sa_locks system procedure” on page 1168
- “How locking works” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement prevents other transactions from modifying the Customers table for the duration of the current transaction:

```
LOCK TABLE GROUPO.Customers
IN SHARE MODE;
```

LOOP statement
Repeats the execution of a statement list.

Syntax

```
[ statement-label : ]
[ WHILE search-condition ] LOOP
statement-list
END LOOP [ statement-label ]
```

Remarks
The WHILE and LOOP statements are control statements that allow you to execute a list of SQL statements repeatedly while a search-condition evaluates to TRUE. The LEAVE statement can be used to resume execution at the first statement after the END LOOP.

If the ending statement-label is specified, it must match the beginning statement-label.
Privileges
None.

Side effects
None.

See also
● “FOR statement” on page 805
● “CONTINUE statement” on page 546
● “WHILE statement [T-SQL]” on page 1050

Standards and compatibility
● SQL/2008  The LOOP/END LOOP statement is part of optional SQL/2008 language feature P002, “Computational completeness”. In SQL/2008, the WHILE DO/END WHILE statement is a separate statement that is also part of language feature P002. The syntax combination WHILE search-condition LOOP supported in SQL Anywhere is a vendor extension.

● Transact-SQL  LOOP is not supported in the Transact-SQL dialect. Looping within Transact-SQL stored procedures is done with the Transact-SQL WHILE statement.

Example
The following example fragment shows a WHILE loop in a procedure.

```sql
... 
SET i = 1;
WHILE i <= 10 LOOP
  INSERT INTO Counters( number ) VALUES ( i );
  SET i = i + 1;
END LOOP;
... 
```

The following example fragment shows a labeled LOOP in a procedure.

```sql
SET i = 1;
lbl:
LOOP
  INSERT INTO Counters( number ) VALUES ( i );
  IF i >= 10 THEN
    LEAVE lbl;
  END IF;
  SET i = i + 1;
END LOOP lbl
```

MERGE statement
Merges tables, views, and procedure results into a table or view.
Syntax

MERGE
INTO target-object [ into-column-list ]
USING [ WITH AUTO NAME ] source-object
  ON merge-search-condition
merge-operation [...] 
  [ OPTION ( query-hint, ... ) ]

target-object : 
  [ userid.]target-table-name [ [ AS ] target-correlation-name ]
  [ userid.]target-view-name [ [ AS ] target-correlation-name ]
  ( select-statement ) [ AS ] target-correlation-name

source-object : 
  [ userid.]source-table-name [ [ AS ] source-correlation-name ] [ WITH ( table-hints ) ]
  [ userid.]source-view-name [ [ AS ] source-correlation-name ]
  [ userid.]source-mat-view-name [ [ AS ] source-correlation-name ]
  ( select-statement ) [ AS ] source-correlation-name [ using-column-list ]

procedure:
  [ owner.]procedure-name ( procedure-syntax )
    [ WITH ( column-name data-type, ... ) ]
    [ [ AS ] source-correlation-name ]

merge-search-condition :
  search-condition
  | PRIMARY KEY

merge-operation :
  WHEN MATCHED [ AND search-condition ] THEN match-action
  | WHEN NOT MATCHED [ AND search-condition ] THEN not-match-action

match-action :
  DELETE
  | RAISERROR [ error-number ]
  | SKIP
  | UPDATE SET set-item, ...
  | UPDATE [ DEFAULTS { ON | OFF } ]

not-match-action :
  INSERT
  | insert-column-list ] VALUES ( value, ... )
  | RAISERROR [ error-number ]
  | SKIP

set-item :
  [ target-correlation-name.]column-name = { expression | DEFAULT }
  [ [ owner-name.]target-table-name.column-name = { expression | DEFAULT }

insert-column-list :
  ( column-name, ... )

query-hint :
  MATERIALIZED VIEW OPTIMIZATION option-value
FORCE OPTIMIZATION  

\| option-name = option-value 

\| into-column-list :  
( column-name, ... ) 

\| using-column-list :  
( column-name, ... ) 

\| error-number : positive integer or variable greater than 17000 

\| option-name : identifier 

\| option-value :  
\| hostvar (indicator allowed) 
\| string 
\| identifier 
\| number 

\| table-hints : See “FROM clause” on page 810. 

\| search-condition : See “Search conditions” on page 40. 

\| set-clause-list : See “SET statement” on page 985. 

Parameters 

\| INTO clause Use this clause to define the target object for the MERGE statement. target-object can be the name of a base table, regular view, or derived table; it cannot be a materialized view. The derived table or view must represent an updatable query block. For example, if the view or derived table definition contains UNION, INTERSECT, EXCEPT, or GROUP BY, then it cannot be used as a target object for the MERGE statement. 

When target-object is a derived table, the optional into-column-list can be used to provide alternate names for the columns of the derived table. When used in this manner, the size of the into-column-list must match the column list for the derived table, and the ordering of the two lists must be the same. 

When target-object is a base table or view, into-column-list can be used to specify a subset of the table or view columns as relevant for the rest of the MERGE statement. 

The database server uses into-column-list to resolve: 

\| UPDATE without a SET clause in WHEN MATCHED clause 
\| INSERT without a VALUES clause in a WHEN NOT MATCHED clause 
\| PRIMARY KEY search condition in the ON clause 
\| WITH AUTO NAME clause in the USING clause 

If you do not specify into-column-list, then into-column-list is assumed to contain all the columns of the target-object. 

\| USING clause Use this clause to define the source of the data you are merging from. source-object can be a base table (including table hints), a view, a materialized view, a derived table, or a procedure. If
source-object is a derived table, you can specify using-column-list. All columns of source-object are used if you do not specify using-column-list.

**WITH AUTO NAME clause** Use this clause to get the server to automatically use column names to match columns in the into-column-list columns in target-object for the merge operation. The following examples are equivalent and demonstrate that the order of the columns in into-column-list changes to match the names of the columns in the source-object when WITH AUTO Name is specified:

```sql
... INTO T ( Name, ID, Description )
   USING WITH AUTO NAME ( SELECT Description, Name, ID FROM PRODUCTS WHERE Description LIKE '%cap%' )
... INTO T ( Description, Name, ID )
   USING ( SELECT Description, Name, ID FROM PRODUCTS WHERE Description LIKE '%cap%' )
```

**ON clause** Use this clause to specify the condition to match a row in source-object with rows in target-object.

You can specify ON PRIMARY KEY to match source-object rows based on the target-object primary key definition. source-object does not need a primary key. However, target-object must have a primary key. When specifying ON PRIMARY KEY:

- An error is returned if target-object is not a base table, or if it does not have a primary key.
- An error is returned if one or more primary key columns are not included in into-column-list.
- The number of columns in into-column-list and using-column-list can be different as long as every primary key column in into-column-list has a corresponding matching column in using-column-list. For example, if into-column-list is (I1, I2, I3), using-column-list is (U1, U2), and the primary key columns are (I2, I3), an error is returned because column (I3) of the target-object primary key does not have a match in the using-column-list.
- Regardless of the definition of the primary key, matching of primary key columns in into-column-list to expressions in using-column-list is based on the position of the primary key columns in into-column-list. For example, suppose the primary key on target-object is defined as (B, C), and the into-column-list is (E, C, F, A, D, B). When ON PRIMARY KEY is specified, target-object column B is compared to the sixth element of using-column-list because column B is in the sixth position in the into-column-list. Likewise, target-object column C is compared to the second element of using-column-list.

ON PRIMARY KEY is syntactic shorthand for a corresponding ON condition. For example, assume that into-column-list is (I1, I2, .. In), and that the corresponding matched using-column-list is (U1, U2, .. Um). Also assume that the primary key columns of target-object are I1, I2, I3 and all the primary key columns are contained in into-column-list. In this case, merge-search-condition is defined as the conjunct "I1=U1 AND I2=U2 AND I3=U3".

**WHEN MATCHED and WHEN NOT MATCHED clauses** Use the WHEN MATCHED and WHEN NOT MATCHED clauses to define an action to take when a row from source-object matches or does not match a row in target-object. You specify the action after the THEN keyword. You can control the actions to take for subsets of matching or non-matching rows by specifying an additional AND clause.

The ON clause determines how rows from source-object are separated into matching and non-matching rows. A row in source-object is considered a matching row when the ON clause is TRUE for at least one
row in target-object. A row from source-object is considered a non-matching row when the ON clause is not TRUE for any rows in target-object. Use multiple WHEN MATCHED and WHEN NOT MATCHED clauses to partition sets of matching and non-matching rows into disjoint subsets. Each subset is processed by a WHEN clause. WHEN MATCHED and WHEN NOT MATCHED clauses are processed in the order they appear in the MERGE statement.

The search condition specified in the AND clause of a WHEN MATCHED or WHEN NOT MATCHED clause determines if a candidate row is processed by the specific clause. When you specify a WHEN MATCHED or WHEN NOT MATCHED clause without the AND clause the search condition in the AND clause is assumed to be TRUE. If a row satisfies the AND condition for more than one clause, the row is processed by the clause that appears first in the MERGE statement.

An error is returned when any of the WHEN MATCHED clauses process the same target-object row more than once. A target-object row can belong to the same subset of the same WHEN MATCHED clause more than once if it matches two different input rows from the source-object.

In the following example an error is returned because the row with ID 300 from the target-object Products matches 111 rows from the source-object SalesOrderItems. All the matches belong to the same subset corresponding to the WHEN MATCHED THEN UPDATE clause.

```
MERGE INTO GROUPO.Products
USING GROUPO.SalesOrderItems S
ON S.ProductID = Products.ID
WHEN MATCHED THEN UPDATE SET Products.Quantity = 20;
```

**WHEN MATCHED**: For a matching row, you can specify one of the following actions for match-action:

- **DELETE** Specify DELETE to delete the row from target-object.

- **RAISERROR** Specify RAISERROR to terminate the merge operation, roll back any changes, and return an error. By default, when you specify RAISERROR, the database server returns SQLSTATE 23510 and SQLCODE -1254.

  Optionally, you can customize the SQLCODE that is returned by specifying the `error-number` parameter after the RAISERROR keyword. The custom SQLCODE must be a positive integer greater than 17000, and can be specified either as a number or a variable. When you specify a custom SQLCODE, the number returned is a negative number.

  For example, if you specify WHEN MATCHED AND `search-condition` THEN RAISERROR 17001, then, when a row is found that satisfies the conditions of the WHEN clause, the merge operation fails, changes are rolled back, and the error returned has SQLSTATE 23510 and SQLCODE -17001.

- **SKIP** Specify SKIP to skip the row; no action is taken.

- **UPDATE** Specify UPDATE SET to update the row using the `set-item` values. `set-item` is a simple assignment expression where a column is set to the value of `expression`. There are no restrictions on the `expression`. You can also specify DEFAULT to set the column to the default defined for the column.
For example, UPDATE SET target-column1=DEFAULT, target-column2=source-column2 sets target-column1 to its default value and sets target-column2 to be the same as the modify row from source-column2 in source-object.

If you do not specify the SET clause, the SET clause is defined by into-column-list and using-column-list. For example, if into-column-list is (I1, I2, .. In), and using-column-list is (U1, U2, .. Un) the SET clause is assumed to be "SET I1=U1 , I2=U2 , .. In=Un".

**WHEN NOT MATCHED**: For a non-matching row, you can specify one of the following actions for non-match-action:

- **INSERT** Specify INSERT...VALUES to insert the row using the specified values. When you specify the INSERT clause without a VALUES clause, the VALUES clause is defined by into-column-list and using-column-list. For example, if into-column-list is (I1, I2, .. In), and using-column-list is (U1, U2, .. Un), the INSERT without a VALUES clause is equivalent to INSERT (I1, I2, .. In) VALUES (U1, U2, .. Un).

- **RAISERROR** Specify RAISERROR to terminate the merge operation, roll back any changes, and return an error. When you specify RAISERROR, the database server returns SQLSTATE 23510 and SQLCODE -1254 by default. Optionally, you can customize the SQLCODE that is returned by specifying the error-number parameter after the RAISERROR keyword. The custom SQLCODE must be a positive integer greater than 17000, and can be specified either as a number or a variable. When you specify a custom SQLCODE, the number returned is a negative number.

  For example, if you specify WHEN NOT MATCHED AND search-condition THEN RAISERROR 17001, then, when a row is found that satisfies the conditions of the WHEN clause, the merge operation fails, changes are rolled back, and the error returned has SQLSTATE 23510 and SQLCODE -17001.

- **SKIP** Specify SKIP to skip the row; no action is taken.

**OPTION clause** Use this clause to specify hints for executing the statement. The following hints are supported:

- **MATERIALIZED VIEW OPTIMIZATION option-value**
- **FORCE OPTIMIZATION**
- **option-name = option-value. A OPTION( isolation_level = ... ) specification in the query text overrides all other means of specifying isolation level for a query.**

**Remarks**

Rows in source-object are compared to rows in target-object and found to be matching or non-matching depending on whether they satisfy the conditions of the ON clause. Rows in source-object are considered a match if there exists at least one row in target-table such that merge-search-condition evaluates to true. Matching rows and non-matching rows are then grouped by the actions defined for them in the WHEN MATCHED and WHEN NOT MATCHED clauses according to the search conditions specified by the AND clauses. The process of grouping rows by matched and non-matched actions is referred to as branching, and each group is referred to as a branch.

Once branching is complete, the database begins executing the action defined for the rows of the branch. Branches are processed in the order in which they occur, which matches the order in which the WHEN
clauses occur in the statement. If, during branching, more than one row in source-object has an action defined for the same row in target-object, the merge operation fails and an error is returned. This prevents the merge operation from performing more than one action on any given row in target-object.

As branches are processed, the insert, update, and delete actions are recorded in the transaction log as their respective INSERT, UPDATE, and DELETE statements.

Privileges

Required privileges are determined at execution time and depend on the objects specified, and the operations the merge would result in:

- **Selecting source-object or target-object** You must be the owner of the objects, or have the SELECT ANY TABLE system privilege, or have SELECT privilege on the target object.

- **Inserting rows into target-object** You must be the owner of target-object, or have the INSERT ANY TABLE system privilege, or have INSERT privilege on the target object.

- **Updating rows in target-object** You must be the owner of target-object, or have the UPDATE ANY TABLE system privilege, or have UPDATE privilege on the target object.

- **Deleting rows from target-object** You must be the owner of target-object, or have the DELETE ANY TABLE system privilege, or have DELETE privilege on the target object.

EXECUTE privilege is required on any procedure referenced in the MERGE statement.

Side effects

Any triggers defined for target-object are fired.

See also

- “Data import with the MERGE statement” [SQL Anywhere Server - SQL Usage]
- “UPDATE statement” on page 1037
- “INSERT statement” on page 860
- “DELETE statement” on page 741
- “SELECT statement” on page 955
- “Search conditions” on page 40
- “MERGE statement for table ‘%1’ failed because of a RAISERROR specification in the statement” [Error Messages]
- “Using the RAISERROR action” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- **SQL/2008** The MERGE statement comprises features F312 and F313 of the SQL/2008 standard. The MERGE statement in SQL Anywhere is compliant with the MERGE statement specification in the SQL/2008 standard, with additional extensions. The SQL Anywhere-specific extensions to the MERGE statement include:
  
  - DELETE in a WHEN MATCHED clause
  - RAISERROR in a WHEN [NOT] MATCHED clause
○ SKIP in a WHEN [NOT] MATCHED clause
○ OPTION clause
○ PRIMARY KEY clause
○ DEFAULTS clause
○ INSERT clause without a VALUES clause
○ WITH AUTO NAME clause
○ UPDATE clause without the SET clause

Examples
The following example merges a row from a derived table into the Products table, effectively adding a new tee shirt with the same attributes as an existing tee shirt, but with a new color, quantity, and product identifier. In this example if the product with identification number 304 already exists in the Products table then the row is not inserted:

```
MERGE INTO Products ( ID, Name, Description, Size, Color, Quantity,
                      UnitPrice, Photo )
USING WITH AUTO NAME ( 
    SELECT 304 AS ID,
       'Purple' AS Color,
        100 AS Quantity,
          Name,
       Description,
         Size,
     UnitPrice,
       Photo
       FROM Products WHERE Products.ID = 300 ) AS DT
ON PRIMARY KEY
WHEN NOT MATCHED THEN INSERT;
```

MESSAGE statement
Displays a message.

Syntax
```
MESSAGE expression
[ TYPE { INFO | ACTION | WARNING | STATUS } ]
[ TO { CONSOLE |
     CLIENT [ FOR { CONNECTION conn-id-number [ IMMEDIATE ] | ALL } ] |
     [ EVENT | SYSTEM ] LOG } |
     [ DEBUG ONLY ] ]
```

```
conn-id : integer
```

Parameters

**TYPE clause**  This clause specifies the message type. The client application must decide how to handle the message. For example, if you specify TO CLIENT, Interactive SQL displays messages in the following locations:
INFO  The Messages tab. INFO is the default type.

ACTION  A window with an OK button.

WARNING  A window with an OK button.

STATUS  The Messages tab.

TO clause  This clause specifies the destination of a message:

CONSOLE  Send messages to the database server messages window and the database server message log file if one has been specified. CONSOLE is the default.

CLIENT  Send messages to the client application. Your application must decide how to handle the message, and you can use the TYPE as information on which to base that decision.

LOG  Send messages to the database server message log file specified by the -o option. If EVENT or SYSTEM is specified, the message is also written to the database server messages window and to the Windows event log under event source SQLANY 16.0 Admin and to the Unix Syslog under the name SQLANY 16.0 Admin (servername) on 32-bit database servers. On 64-bit database servers, SQLANY64 16.0 Admin is the event source. Messages in the database server message log are identified as follows:

- i  Messages of type INFO or STATUS.
- w  Messages of type WARNING.
- e  Messages of type ACTION.

FOR clause  For messages TO CLIENT, this clause specifies which connections receive notification about the message. By default, the connection receives the message the next time a SQL statement or a WAITFOR DELAY statement is executed.

CONNECTION conn-id-number  Specify the recipient's connection ID number. If IMMEDIATE is specified, the connection receives the message within a few seconds regardless of when the SQL statement is executed.

ALL  Specify that all open connections receive the message.

DEBUG ONLY  This clause allows you to control whether debugging messages added to stored procedures and triggers are enabled or disabled by changing the setting of the debug_messages option. When DEBUG ONLY is specified, the MESSAGE statement is executed only when the debug_messages option is set to On.

Note  DEBUG ONLY messages are inexpensive when the debug_messages option is set to Off, so these statements can usually be left in stored procedures on a production system. However, they should be used sparingly in locations where they would be executed frequently; otherwise, they may result in a small performance penalty.
Remarks

The MESSAGE statement displays a message, which can be any expression. Clauses can specify the message type and where the message appears.

If the size of expression exceeds the database page size, expression is truncated to fit within the database page size. To check the page size in effect for the database, you can query the PageSize database property (SELECT DB_PROPERTY( 'PageSize' );).

The procedure executing a MESSAGE...TO CLIENT statement must be associated with a connection.

For example, the window is not displayed in the following example because the event occurs outside a connection.

```sql
CREATE EVENT CheckIdleTime
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
  MESSAGE 'Idle server' TYPE WARNING TO CLIENT;
END;
```

However, in the following example, the message is written to the database server messages window.

```sql
CREATE EVENT CheckIdleTime2
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
  MESSAGE 'Idle server' TYPE WARNING TO CONSOLE;
END;
```

Valid expressions can include a quoted string or other constant, variable, or function.

The FOR clause can be used to notify another application of an event detected on the database server without the need for the application to explicitly check for the event. When the FOR clause is used, recipients receive the message the next time that they execute a SQL statement. If the recipient is currently executing a SQL statement, the message is received when the statement completes. If the statement being executed is a stored procedure call, the message is received before the call is completed.

If an application requires notification within a short time after the message is sent and when the connection is not executing SQL statements, use the IMMEDIATE clause to implement client notification and not multiple concurrent WAITFOR DELAY statements.

Typically, messages sent using the IMMEDIATE clause are delivered in less than five seconds, even if the destination connection is not making database server requests. Message delivery could be delayed if the client connection makes several requests per second, receives very large BLOB data, or if the client's message callback executes for more than a second. In addition, sending more than one IMMEDIATE message to a single connection every two seconds could delay message delivery or generate an error message. If the client connection is disconnected, a successful MESSAGE...IMMEDIATE statement may not be delivered.

Messages sent without the IMMEDIATE clause are only delivered when the client executes a specific request, or a WAITFOR DELAY statement. As a result, the delivery time of messages is unlimited.
The IMMEDIATE clause requires a SQL Anywhere 11 or later DBLib, ODBC, or SQL Anywhere JDBC driver. The IMMEDIATE clause is not supported by non-threaded Unix client libraries. An error message is generated when a message is sent to a destination connection that does not support the IMMEDIATE clause. An error message is generated when an IMMEDIATE message is sent to the same connection executing the MESSAGE statement.

```
MESSAGE 'Please disconnect' TYPE WARNING TO CLIENT
FOR CONNECTION 16 IMMEDIATE;
```

A MESSAGE...TO CLIENT expression can be truncated to 2048 bytes. For messages sent with the IMMEDIATE clause, the message expression can be truncated to the smaller of the packet size of the connection or 2048 bytes.

Embedded SQL and ODBC clients receive messages via message callback functions. In each case, these functions must be registered. In embedded SQL, the message callback is registered with `db_register_a_callback` using the `DB_CALLBACK_MESSAGE` parameter. In ODBC, the message callback is registered with `SQLSetConnectAttr` using the `SA_REGISTER_MESSAGE_CALLBACK` parameter.

### Privileges

To execute a MESSAGE statement containing a FOR clause, a TO EVENT LOG clause, or a TO SYSTEM LOG clause, you must have the SERVER OPERATOR system privilege. Otherwise, no privileges are required for this statement.

### Side effects

None.

### See also

- “sa_conn_info system procedure” on page 1106
- “CREATE PROCEDURE statement” on page 639
- “debug_messages option” [SQL Anywhere Server - Database Administration]
- “db_register_a_callback function” [SQL Anywhere Server - Programming]
- “WAITFOR statement” on page 1047

### Standards and compatibility

- **SQL/2008** Vendor extension.

### Example

The following procedure displays a message in the database server messages window:

```
CREATE PROCEDURE message_text( )
BEGIN
  MESSAGE 'The current date and time: ', Now( );
END;
```

The following statement displays the string The current date and time, followed by the current date and time, in the database server messages window.

```
CALL message_text( );
```
NOTIFY TRACE EVENT statement

Logs a user-defined trace event to a trace session.

Syntax

```
NOTIFY TRACE EVENT trace-event-name ( [ param1 [ ,... ] ]
```

Parameters

- `trace-event-name` The trace event name must be the name of a user-defined trace event. It cannot be a system-defined trace event.
- `param1` The values of the trace event fields.

Remarks

This statement is used to notify any sessions that include the specified trace event. If a trace event is not being traced by any session, then this statement has no effect and the parameters are not evaluated (for example, by a call to a user-defined function).

System privileges

You must have the NOTIFY TRACE EVENT system privilege.

Side effects

None

See also

- “Event tracing” [SQL Anywhere Server - Database Administration]
- “CREATE TEMPORARY TRACE EVENT statement” on page 705
- “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
- “ALTER TRACE EVENT SESSION statement” on page 506
- “DROP TRACE EVENT statement” on page 783
- “DROP TRACE EVENT SESSION statement” on page 784
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_fields system procedure” on page 1312
- “sp_trace_event_sessions system procedure” on page 1317
- “sp_trace_event_session_events system procedure” on page 1314
- “sp_trace_event_session_targets system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statements log events to the current (fictitious) event trace session, my_event.
OPEN statement [ESQL] [SP]

Opens a previously declared cursor to access information from the database.

Syntax 1 [ESQL]

```
OPEN cursor-name
  [ USING { DESCRIPTOR sqlda-name | hostvar, ... } ]
  [ WITH HOLD ]
  [ ISOLATION LEVEL isolation-level ]
  [ BLOCK n ]
```

Syntax 2 [SP]

```
OPEN cursor-name
  [ WITH HOLD ]
  [ ISOLATION LEVEL isolation-level ]

cursor-name : identifier | hostvar

sqlda-name : identifier

isolation-level : 0 | 1 | 2 | 3 | SNAPSHOT | STATEMENT SNAPSHOT | READONLY STATEMENT

SNAPSHOT
```

Parameters

**USING DESCRIPTOR clause** The USING DESCRIPTOR clause is for embedded SQL only. It specifies the host variables to be bound to the placeholder bind variables in the SELECT statement for which the cursor has been declared.

OPEN...USING cannot be used in a stored procedure.

**WITH HOLD clause** By default, all cursors are automatically closed at the end of the current transaction (COMMIT or ROLLBACK). The optional WITH HOLD clause keeps the cursor open for subsequent transactions. It remains open until the end of the current connection or until an explicit CLOSE statement is executed. Cursors are automatically closed when a connection is terminated.

Upon COMMIT or ROLLBACK, all long-term row locks held by the connection are released, including those rows that constitute the result set of a WITH HOLD cursor. However, cursor stability locks, which are acquired at isolation levels 1, 2, and 3, are retained for the life of the cursor and are only released when the cursor is closed or when the connection terminates.

Upon completion of a ROLLBACK statement, the contents of, and positioning within, a WITH HOLD cursor are unpredictable and are not guaranteed. You can use the ansi_close_cursors_on_rollback option to control whether or not a ROLLBACK statement will close WITH HOLD cursors automatically.

**ISOLATION LEVEL clause** The ISOLATION LEVEL clause allows this cursor to be opened at an isolation level different from the current setting of the isolation_level option. All operations on this cursor
are performed at the specified isolation level regardless of the option setting. If this clause is not specified, then the cursor's isolation level for the entire time the cursor is open is the value of the isolation_level option when the cursor is opened.

The following values are supported:

○ 0
○ 1
○ 2
○ 3
○ SNAPSHOT
○ STATEMENT SNAPSHOT
○ READONLY STATEMENT SNAPSHOT

The cursor is positioned before the first row.

**BLOCK clause**  This clause is for embedded SQL use only. Rows may be fetched by the client application more than one at a time. This is referred to as block fetching, prefetching, or multi-row fetching. The BLOCK clause can reduce the number of rows prefetched. Specifying the BLOCK clause on OPEN is the same as specifying the BLOCK clause on each FETCH.

**Remarks**

The OPEN statement opens the named cursor. The cursor must be previously declared.

The OPEN statement may return a SQL warning if the cursor type does not match the characteristics of the cursor's underlying statement.

When the cursor is on a CALL statement, OPEN causes the procedure to execute until the first result set (SELECT statement with no INTO clause) is encountered. If the procedure completes and no result set is found, the SQLSTATE_PROCEDURE_COMPLETE warning is set.

- **Embedded SQL usage**  After successful execution of the OPEN statement, the `sqlerrd[3]` field of the SQLCA (SQLIOESTIMATE) is filled in with an estimate of the number of input/output operations required to fetch all rows of the query. Also, the `sqlerrd[2]` field of the SQLCA (SQLCOUNT) is filled with either the actual number of rows in the cursor (a value greater than or equal to 0), or an estimate thereof (a negative number whose absolute value is the estimate). It is the actual number of rows if the database server can compute it without counting the rows. The database can also be configured to always return the actual number of rows, but this can be expensive.

If `cursor-name` is specified by an identifier or string, the corresponding DECLARE CURSOR statement must appear before the OPEN in the C program; if the `cursor-name` is specified by a host variable, the DECLARE CURSOR statement must execute before the OPEN statement.

**Privileges**

When the cursor is on a SELECT statement, you must be the owner of the object referenced in the cursor, or have SELECT privilege on the object, or have the appropriate SELECT system privilege (for example, SELECT ANY TABLE).

When the cursor is on a CALL statement, you must be the owner of the procedure or have EXECUTE privilege on the procedure, or have the EXECUTE ANY PROCEDURE system privilege.
Side effects

OPEN causes the complete materialization of an INSENSITIVE cursor's result set.

If access plan caching is enabled, some SQL warnings that would be returned to the application at OPEN time may be suppressed. The suppressed warnings include warnings to indicate that the cursor type has changed, that the underlying query is not deterministic, or that string truncation has occurred with one or more literal constants embedded in the statement.

See also

- “DECLARE CURSOR statement [ESQL] [SP]” on page 729
- “RESUME statement” on page 939
- “PREPARE statement [ESQL]” on page 915
- “FETCH statement [ESQL] [SP]” on page 800
- “DECLARE CURSOR statement [ESQL] [SP]” on page 729
- “RESUME statement” on page 939
- “CLOSE statement [ESQL] [SP]” on page 537
- “FOR statement” on page 805
- “ansi_close_cursors_on_rollback option” [SQL Anywhere Server - Database Administration]
- “Plan caching” [SQL Anywhere Server - SQL Usage]
- “Insensitive cursors” [SQL Anywhere Server - Programming]
- “row_counts option” [SQL Anywhere Server - Database Administration]
- “close_on_endtrans option” [SQL Anywhere Server - Database Administration]
- “Cursors in embedded SQL” [SQL Anywhere Server - Programming]
- “Cursors in procedures, triggers, user-defined functions, and batches” [SQL Anywhere Server - SQL Usage]
- “How locking works” [SQL Anywhere Server - SQL Usage]
- “Lock duration” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- **SQL/2008** Use of the OPEN statement within embedded SQL is part of optional SQL language feature B031, "Basic dynamic SQL". The use of the OPEN statement within a stored procedure is a core feature of SQL/2008. The ISOLATION LEVEL and BLOCK clauses are vendor extensions, as is the ability to OPEN a cursor over a CALL statement. In the SQL/2008 standard, WITH HOLD is specified as part of the DECLARE CURSOR statement, and not on OPEN.

  The setting of specific values in the SQLCA is a vendor extension.

- **Transact-SQL** The OPEN statement is supported by Adaptive Server Enterprise. Adaptive Server Enterprise does not support the ISOLATION LEVEL, BLOCK, and WITH HOLD clauses.

Example

The following examples show the use of OPEN in embedded SQL.

```sql
EXEC SQL OPEN employee_cursor;
EXEC SQL PREPARE emp_stat FROM
'SELECT empnum, empname FROM GROUPO.Employees WHERE name like ?';
EXEC SQL DECLARE employee_cursor CURSOR FOR emp_stat;
EXEC SQL OPEN employee_cursor USING :pattern;
```
This example fragment shows an OPEN statement in a procedure or trigger.

```
BEGIN
  DECLARE cur_employee CURSOR FOR
  SELECT Surname
  FROM GROUPO.Employees;
  DECLARE name CHAR(40);
  OPEN cur_employee;
  LP: LOOP
    FETCH NEXT cur_employee INTO name;
    IF SQLCODE <> 0 THEN LEAVE LP END IF;
    ...
  END LOOP
  CLOSE cur_employee;
END
```

**OUTPUT statement [Interactive SQL]**

Outputs the current query results to a file or ODBC data source.

**Syntax 1 - Output to a file**
```
OUTPUT TO filename
  [APPEND ]
  [BYTE ORDER MARK { ON | OFF }]
  [COLUMN WIDTHS ( integer, ... )]
  [DELIMITED BY string ]
  [ENCODING encoding ]
  [ESCAPE CHARACTER character ]
  [ESCAPES { ON | OFF }]
  [FORMAT output-format ]
  [HEXADECIMAL { ON | OFF | ASIS }]
  [QUOTE string [ ALL ]]
  [VERBOSTE ]
  [WITH COLUMN NAMES ]
```

```
output-format :
  TEXT
  | FIXED
  | HTML
  | SQL
  | XML
```

```
encoding : string | identifier
```

**Syntax 2 - Output to an ODBC data source**
```
OUTPUT
  USING connection-string
  INTO destination-table-name
  [ CREATE TABLE { ON | OFF } ]
```

```
connection-string :
  { DSN = odbc-data-source
    | DRIVER = odbc-driver-name [ ; connection-parameter = value [ ; ... ] ] }
```
Parameters

**APPEND clause**  This optional keyword is used to append the results of the query to the end of an existing output file without overwriting the previous contents of the file. If the APPEND clause is not used, the OUTPUT statement overwrites the contents of the output file by default. The APPEND keyword is valid if the output format is TEXT, FIXED, or SQL.

**BYTE ORDER MARK clause**  Use this clause to specify whether to include a byte order mark (BOM) at the start of a Unicode file. By default, this option is ON, which directs Interactive SQL to write a byte order mark (BOM) at the beginning of the file. If BYTE ORDER MARK is OFF, DBISQL does not write a BOM.

The BYTE ORDER MARK clause is relevant only when writing TEXT formatted files. Attempts to use the BYTE ORDER MARK clause with FORMAT clauses other than TEXT returns an error.

The BYTE ORDER MARK clause is used only when reading or writing files encoded with UTF-8 or UTF-16 (and their variants). Attempts to use the BYTE ORDER MARK clause with any other encoding returns an error.

**COLUMN WIDTHS clause**  The COLUMN WIDTHS clause is used to specify the column widths for the FIXED format output.

**CREATE TABLE clause**  Use the CREATE TABLE clause to specify whether to create the destination table if it does not exist. The default is ON.

**DELIMITED BY clause**  The DELIMITED BY clause is for the TEXT output format only. The delimiter string is placed between columns. The default is comma.

**ENCODING clause**  The ENCODING clause allows you to specify the encoding that is used to write the file. The ENCODING clause can only be used with the TEXT format.

The ENCODING clause is useful when you have data that cannot be represented in the operating system character set. In this case, if you do not use the ENCODING clause, characters that cannot be represented in the default encoding are lost in the output (that is, a lossy conversion occurs).

If the input file was created using the OUTPUT statement and an encoding was specified, then the same ENCODING clause should be specified on the INPUT statement.

When running Interactive SQL, the encoding that is used to export the data is determined in the following order:

- The encoding specified by the ENCODING clause (if this clause is specified)
- The encoding specified with the default_isql_encoding option (if this option is set).
- The default encoding for the platform you are running on. On English Windows computers, the default encoding is 1252.

**ESCAPE CHARACTER clause**  The default escape character for characters stored as hexadecimal codes and symbols is a backslash (\). For example, \x0A is the linefeed character.
This setting can be changed using the ESCAPE CHARACTER clause. For example, to use the
exclamation mark as the escape character, specify:

```
... ESCAPE CHARACTER '!' 
```

The new line character can be specified as '\n'. Other characters can be specified using hexadecimal
ASCII codes, such as \x09 for the tab character. A sequence of two backslash characters (\\) is
interpreted as a single backslash. A backslash followed by any character other than n, x, X, or \ is
interpreted as two separate characters. For example, q is interpreted as a backslash and the letter q.

**ESCAPES clause**  With ESCAPES turned on (the default), characters following the backslash
character are recognized and interpreted as special characters by the database server. With ESCAPES
turned off, the characters are written exactly as they appear in the source data.

**FORMAT clause**  The FORMAT clause allows you to specify the file format for the output. If you do
not specify the FORMAT clause, the format specified by the output_format option is used. If you specify
the FORMAT clause, the setting of the output_format option is ignored. The default output format is
TEXT. Allowable output formats are:

- **TEXT**  The output is a TEXT format file with one row per line in the file. All values are separated
  by commas, and strings are enclosed in apostrophes (single quotes). The delimiter and quote strings
can be changed using the DELIMITED BY and QUOTE clauses. If ALL is specified in the QUOTE
  clause, all values (not just strings) are quoted. TEXT is the default output type.

  Three other special sequences are also used. The two characters \n represent a newline character, \ \represents a single \
, and the sequence \xDD represents the character with hexadecimal code DD.

  To TEXT without including quotes or newlines in your output, turn off quotes and escapes as follows:
  `QUOTE '' ESCAPES OFF`.

- **FIXED**  The output is fixed format with each column having a fixed width. The width for each
  column can be specified using the COLUMN WIDTHS clause. No column headings are output in this
  format.

  If the COLUMN WIDTHS clause is omitted, the width for each column is computed from the data
type for the column, and is large enough to hold any value of that data type. The exception is that
LONG VARCHAR and LONG BINARY data default to 32 KB.

- **HTML**  The output is in the Hyper Text Markup Language format.

- **SQL**  The output is an Interactive SQL INPUT statement (required to recreate the information in the
  table) in a `.sql` file.

- **XML**  The output is an XML file encoded in UTF-8 and containing an embedded DTD. Binary
  values are encoded in CDATA blocks with the binary data rendered as 2-hex-digit strings.

**HEXADECIMAL clause**  The HEXADECIMAL clause specifies how binary values are output for the
TEXT format. Allowable values are:

- **ON**  When set to ON, binary values are written with an Ox prefix followed by a series of
  hexadecimal pairs; for example, 0xabcd.
- **OFF**  When set to OFF, unprintable character values are prefixed with the escape character, such as a backslash, followed by an x, and then followed by the hexadecimal pair for the byte. Printable characters are output as-is.

- **ASIS**  When set to ASIS, values are written as is, without any escaping, even if the values contain control characters. ASIS is useful for text that contains formatting characters such as tabs or carriage returns.

**QUOTE clause**  The QUOTE clause is for the TEXT output format only. The quote string is placed around string values. The default is a single quote ('). If ALL is specified in the QUOTE clause, the quote string is placed around all values, not just around strings. To suppress quoting, specify empty single quotes. For example, `QUOTE ''`.

**USING clause**  The USING clause exports data to an ODBC data source. You can either specify the ODBC data source name with the DSN option, or the ODBC driver name and connection parameters with the DRIVER option. *Connection-parameter* is an optional list of database-specific connection parameters.

`odbc-data-source` is the name of a user or ODBC data source name. For example, `odbc-data-source` for the SQL Anywhere sample database is SQL Anywhere 16 Demo.

`odbc-driver-name` is the ODBC driver name. For a SQL Anywhere database, the `odbc-driver-name` is SQL Anywhere; for an UltraLite database, `odbc-driver-name` is UltraLite 16.

**VERBOSE clause**  When the optional VERBOSE keyword is included, error messages about the query, the SQL statement used to select the data, and the data itself are written to the output file. Lines that do not contain data are prefixed by two hyphens. If VERBOSE is omitted (the default) only the data is written to the file. The VERBOSE keyword is valid if the output format is TEXT, FIXED, or SQL.

**WITH COLUMN NAMES clause**  The WITH COLUMN NAMES clause inserts the column names in the first line of the text file. The WITH COLUMN NAMES clause is for TEXT format only.

**Remarks**

The OUTPUT statement outputs data to a file or database. The OUTPUT statement is used directly after a statement that retrieves the data to be output.

To export multiple result sets, use syntax 1 and set the isql_show_multiple_result_sets option to On. Interactive SQL creates a file for each result set. The files are named `filename-x`, where `x` is a counter starting at 1. For example, specifying OUTPUT TO a file named `data.txt` results in files named `data-1.txt`, `data-2.txt`, and so on.

You cannot use syntax 2 to export multiple result sets.

The output format can be specified with the optional FORMAT clause. The default format is TEXT. If no FORMAT clause is specified, the Interactive SQL output_format option setting is used.

Because the OUTPUT statement is an Interactive SQL command, it cannot be used in any compound statement (such as IF), or in a stored procedure.

When exporting columns containing BINARY or LONG BINARY data to a Microsoft Excel workbook, convert the data to a string or number. In addition, when data is exported to a Microsoft Excel workbook,
the data is read-only unless the ReadOnly parameter is set to zero or turned off when the DSN option is selected.

Privileges

None.

Side effects

In Interactive SQL, the Results tab displays the results of the current query.

See also

- “SELECT statement” on page 955
- “INPUT statement [Interactive SQL]” on page 854
- “UNLOAD statement” on page 1027
- “output_format option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “Data import and export” [SQL Anywhere Server - SQL Usage]
- “default_isql_encoding option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “isql_show_multiple_result_sets option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “Interactive SQL” [SQL Anywhere Server - Database Administration]
- “Tips on exporting data with the OUTPUT statement” [SQL Anywhere Server - SQL Usage]
- “Statements allowed in procedures, triggers, events, and batches” [SQL Anywhere Server - SQL Usage]
- “Returning multiple result sets” [SQL Anywhere Server - SQL Usage]
- “output_format option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “Supported character sets” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Examples

Place the contents of the Employees table in a text file:

```sql
SELECT * FROM GROUPO.Employees;
OUTPUT TO 'c:\temp\Employees.txt'
FORMAT TEXT;
```

Place the contents of the Employees table at the end of an existing text file, and include any messages about the query in this file as well:

```sql
SELECT * FROM GROUPO.Employees;
OUTPUT TO 'c:\temp\Employees.txt'
APPEND VERBOSE;
```

Suppose you need to export a value that contains an embedded line feed character. A line feed character has the numeric value 10, which you can represent as the string 'x0a' in a SQL statement. For example, execute the following statement, with HEXADECIMAL set to ON:

```sql
SELECT CAST ('line1\x0a\line2' AS VARBINARY);
OUTPUT TO 'c:\temp\file.txt' HEXADECIMAL ON;
```
You get a file with one line in it containing: 0x6c696e65310a6c696e6532

But if you execute the same statement with HEXADECIMAL set to OFF, you get:
'line1\x0Aline2'

Finally, if you set HEXADECIMAL to ASIS, you get a file with two lines:

'line1
line2'

You get two lines when you use ASIS because the embedded line feed character has been exported without being converted to a two digit hexadecimal representation, and without being prefixed by anything.

The following example outputs the data from the Customers table to a new table, Customers2:

```
SELECT * FROM Customers;
OUTPUT USING 'DSN=SQL Anywhere 16 Demo'
INTO "Customers2";
```

The following example copies the Customers table from the sample database to a fictitious database called mydatabase.db, using the DRIVER option.

```
SELECT * FROMCustomers;
OUTPUT USING 'DRIVER=SQL Anywhere 16;UID=DBA;PWD=sql;DBF=c:\test\mydatabase.db'
INTO "Customers";
```

The following example copies the Customers table from the SQL Anywhere sample database into a table called Customers in a fictitious UltraLite database, myULDatabase.db, using the DRIVER option.

```
SELECT * FROM Customers;
OUTPUT USING 'DRIVER=UltraLite 16;DBF=c:\test\myULDatabase.udb'
INTO "Customers";
```

The following example copies the Customers table into a fictitious MySQL database called mydatabase, using the DRIVER option.

```
SELECT * FROM GROUPO.Customers;
OUTPUT USING 'DRIVER=MySQL ODBC 5.1 Driver;DATABASE=mydatabase;SERVER=mySQLHost;UID=me;PWD=secret'
INTO "Customers";
```

The following command outputs a file which contains 'one\x0Atwo\x0Athree':

```
SELECT 'one\ntwo\nthree';
OUTPUT TO 'c:\temp\test.txt' HEXADECIMAL OFF;
```

**PARAMETERS statement [Interactive SQL]**

Specifies parameters to an Interactive SQL script file.

**Syntax**

```
PARAMETERS parameter1, parameter2, ...
```
Remarks
The PARAMETERS statement names the parameters for a script file, so that they can be referenced later in the script file.

Parameters are referenced by putting `{parameter1}` into the file where you want the named parameter to be substituted. There must be no spaces between the braces and the parameter name.

If a script file is invoked with less than the required number of parameters, Interactive SQL prompts for values of the missing parameters.

Privileges
None.

Side effects
None.

See also
- “READ statement [Interactive SQL]” on page 922
- “Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following Interactive SQL script file takes two parameters.

```sql
PARAMETERS department_id, file;
SELECT Surname
FROM GROUPO.Employees
WHERE DepartmentID = {department_id}
>{file}.dat;
```

If you save this script in a fictitious file named `c:\temp\test.sql`, you can run it from Interactive SQL using the following command:

```sql
READ 'c:\temp\test.sql' [100] [data]
```

### PASSTHROUGH statement [SQL Remote]
Starts or stops passthrough mode for SQL Remote administration. Syntaxes 1 and 2 start passthrough mode, while syntax 3 stops passthrough mode.

Syntax 1
```
PASSTHROUGH [ ONLY ] FOR userid, ...
```

Syntax 2
```
PASSTHROUGH [ ONLY ] FOR SUBSCRIPTION
TO [ owner. ]publication-name [ ( constant ) ]
```
Syntax 3

PASSTHROUGH STOP

Remarks

In passthrough mode, any SQL statements are executed by the database server, and are also placed into the transaction log to be sent in messages to subscribers. If the ONLY keyword is used to start passthrough mode, the statements are not executed at the server; they are sent to recipients only. When a passthrough session contains calls to stored procedures, the procedures must exist in the server that is issuing the passthrough commands, even if they are not being executed locally at the server. The recipients of the passthrough SQL statements are either a list of user IDs (syntax 1) or all subscribers to a given publication. Passthrough mode may be used to apply changes to a remote database from the consolidated database or send statements from a remote database to the consolidated database.

Syntax 2 sends statements to remote databases whose subscriptions are started, and does not send statements to remote databases whose subscriptions are created and not started.

Syntax 3 stops passthrough mode on the current connection. You must execute the PASSTHROUGH STOP statement on the same connection that initiated the passthrough mode. If you use syntax 1 or 2 to start passthrough mode on a connection and it disconnects before a PASSTHROUGH STOP statement is executed, the disconnect implicitly executes a PASSTHROUGH STOP statement.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

None.

Standards and compatibility

- SQL/2008 Vendor extension.

See also

- “FORWARD TO statement” on page 808
- “UNLOAD statement” on page 1027
- “Connectivity tests” [SQL Anywhere Server - SQL Usage]
- “SQL Remote passthrough mode” [SQL Remote]
- “Start and stop passthrough mode” [SQL Remote]
- “Passthrough mode limitations” [SQL Remote]
- “Changes to avoid on a running system” [SQL Remote]
- “Upgrades and resynchronization” [SQL Remote]
- “Subscription resynchronization” [SQL Remote]

Example

PASSTHROUGH FOR Sam_Singer ;
...
( SQL statements to be executed at the remote database )
...
PASSTHROUGH STOP ;
PREPARE statement [ESQL]

Prepares a statement to be executed later, or defines a cursor.

Syntax

PREPARE statement-name
    FROM statement [ FOR { UPDATE [ cursor-concurrency ] | READ ONLY } ]
    [ DESCRIBE describe-type INTO [ [ SQL ] DESCRIPTOR ] descriptor ]
    [ WITH EXECUTE ]

statement-name : identifier | hostvar

statement : string | hostvar

describe-type :
    [ ALL | BIND VARIABLES | INPUT | OUTPUT | SELECT LIST ]
    [ LONG NAMES [ [ [ OWNER. ]TABLE. ]COLUMN ]
    | WITH VARIABLE RESULT ]

cursor-concurrency :
    BY { VALUES | TIMESTAMP | LOCK }

Parameters

statement-name  The statement name can be an identifier or host variable. However, you should not use an identifier when using multiple SQLCAs. If you do, two prepared statements may have the same statement number, which could cause the wrong statement to be executed or opened. Also, using an identifier for a statement name is not recommended for multithreaded applications where the statement name may be referenced by multiple threads concurrently.

DESCRIBE clause  If DESCRIBE INTO DESCRIPTOR is used, the prepared statement is described into the specified descriptor. The describe type may be any of the describe types allowed in the DESCRIBE statement.

FOR UPDATE | FOR READ ONLY  Defines the cursor updatability if the statement is used by a cursor. A FOR READ ONLY cursor cannot be used in an UPDATE (positioned) or a DELETE (positioned) operation. FOR READ ONLY is the default.

In response to any request for a cursor that specifies FOR UPDATE, SQL Anywhere provides either a value-sensitive cursor or a sensitive cursor. Insensitive and asensitive cursors are not updatable.

BY VALUES | BY TIMESTAMP  The database server utilizes a keyset-driven cursor to enable the application to be informed when rows have been modified or deleted as the result set is scrolled.

BY LOCK clause  The database server acquires intent row locks on fetched rows of the result set. These are long-term locks that are held until the transaction is committed or rolled back.

- WITH EXECUTE clause  If the WITH EXECUTE clause is used, the statement is executed if and only if it is not a CALL or SELECT statement, and it has no host variables. The statement is immediately dropped after a successful execution. If the PREPARE and the DESCRIBE (if any) are successful but the statement cannot be executed, a warning SQLCODE 111, SQLSTATE 01W08 is set, and the statement is not dropped.
The DESCRIBE INTO DESCRIPTOR and WITH EXECUTE clauses may improve performance because they cut down on the required client/server communication.

- **WITH VARIABLE RESULT clause** The WITH VARIABLE RESULT clause is used to describe procedures that may have more than one result set, with different numbers or types of columns.

If WITH VARIABLE RESULT is used, the database server sets the SQLCOUNT value after the describe to one of the following values:

- **0** The result set may change: the procedure call should be described again following each OPEN statement.
- **1** The result set is fixed. No re-describing is required.

**Note**
For compatibility reasons, preparing COMMIT, PREPARE TO COMMIT, and ROLLBACK statements is still supported. However, it is recommended that you do all transaction management operations with static embedded SQL because certain application environments may require it. Also, other embedded SQL systems do not support dynamic transaction management operations.

**Remarks**

The PREPARE statement prepares a SQL statement from the statement and associates the prepared statement with statement-name. This statement name is referenced to execute the statement, or to open a cursor if the statement is a SELECT or CALL statement. The statement-name may be a host variable of type a_sql_statement_number defined in the sqlca.h header file that is automatically included. If an identifier is used for the statement-name, only one statement per module may be prepared with this statement-name.

If a host variable is used for statement-name, it must have the type short int. There is a typedef for this type in sqlca.h called a_sql_statement_number. This type is recognized by the SQL preprocessor and can be used in a DECLARE section. The host variable is defined by the database during the PREPARE statement, and you do not need to initialize it.

**Privileges**

None.

**Side effects**

Any statement previously prepared with the same name is lost.

The statement is dropped after use only if you use WITH EXECUTE and the execution is successful. You should ensure that you DROP the statement after use in other circumstances. If you do not, the memory associated with the statement is not reclaimed.
See also

- “DECLARE CURSOR statement [ESQL] [SP]” on page 729
- “DESCRIBE statement [ESQL]” on page 744
- “OPEN statement [ESQL] [SP]” on page 904
- “EXECUTE statement [ESQL]” on page 794
- “DROP STATEMENT statement [ESQL]” on page 775
- “Statement cannot be executed” [Error Messages]
- “Dynamic SQL statements” [SQL Anywhere Server - Programming]

Standards and compatibility

- SQL/2008  PREPARE is part of optional SQL/2008 language feature B031, "Basic dynamic SQL". The optional FOR UPDATE, FOR READ ONLY, DESCRIBE, and WITH EXECUTE clauses are vendor extensions.

Example

The following statement prepares a simple query:

```sql
EXEC SQL PREPARE employee_statement FROM
  'SELECT Surname FROM Employees';
```

PREPARE TO COMMIT statement

Checks whether a COMMIT can be performed successfully.

Syntax

```
PREPARE TO COMMIT
```

Remarks

The PREPARE TO COMMIT statement tests whether a COMMIT can be performed successfully. The statement will cause an error if a COMMIT is impossible without violating the integrity of the database.

The PREPARE TO COMMIT statement cannot be used in stored procedures, triggers, events, or batches.

Privileges

None.

Side effects

None.

See also

- “COMMIT statement” on page 541
- “ROLLBACK statement” on page 950

Standards and compatibility

- SQL/2008  Vendor extension.
Example
The following sequence of statements leads to an error because of foreign key checking on the Employees table.

```sql
EXECUTE IMMEDIATE
  "SET OPTION wait_for_commit = 'On';"
EXECUTE IMMEDIATE "DELETE FROM Employees
  WHERE EmployeeID = 160"
EXECUTE IMMEDIATE "PREPARE TO COMMIT"
```

The following sequence of statements does not cause an error when the delete statement is executed, even though it causes integrity violations. The PREPARE TO COMMIT statement returns an error.

```sql
SET OPTION wait_for_commit= 'On';
DELETE
  FROM GROUPO.Departments
  WHERE DepartmentID = 100;
PREPARE TO COMMIT;
```

**PRINT statement [T-SQL]**

Returns a message to the client, or display a message in the database server messages window.

**Syntax**

```sql
PRINT format-string [, arg-list ]
```

**Remarks**

The PRINT statement returns a message to the client window if you are connected from an Open Client application or jConnect application. If you are connected from an embedded SQL or ODBC application, the message is displayed in the database server messages window.

The format string can contain place holders for the arguments in the optional argument list. These place holders are of the form %nn!, where nn is an integer between 1 and 20.

**Privileges**

None.

**Side effects**

None.

**See also**

- “MESSAGE statement” on page 899

**Standards and compatibility**

- SQL/2008 Transact-SQL extension.

**Example**

The following statement displays a message:
PRINT 'Display this message';

The following statement illustrates the use of placeholders in the PRINT statement:

```
DECLARE @var1 INT, @var2 INT
SELECT @var1 = 3, @var2 = 5
PRINT 'Variable 1 = %1!, Variable 2 = %2!', @var1, @var2
```

**PUT statement [ESQL]**

Inserts a row into the specified cursor.

**Syntax**

```
PUT cursor-name
   { USING DESCRIPTOR sqlda-name | FROM hostvar-list }
   [ INTO { DESCRIPTOR sqlda-name | hostvar-list } ]
   [ ARRAY :row-count ]
```

- `cursor-name`: identifier | hostvar
- `sqlda-name`: identifier
- `hostvar-list`: may contain indicator variables
- `row-count`: integer | hostvar

**Remarks**

Inserts a row into the named cursor. Values for the columns are taken from the first SQLDA or the host variable list, in a one-to-one correspondence with the columns in the INSERT statement (for an INSERT cursor) or the columns in the SELECT list (for a SELECT cursor).

The PUT statement can be used only on a cursor over an INSERT or SELECT statement that references a single table in the FROM clause, or that references an updatable view consisting of a single base table.

If the sqldata pointer in the SQLDA is the null pointer, no value is specified for that column. If the column has a DEFAULT VALUE associated with it, that is used; otherwise, a NULL value is used.

The second SQLDA or host variable list contains the results of the PUT statement.

The optional ARRAY clause can be used to perform wide puts, which insert more than one row at a time and which may improve performance. The integer value is the number of rows to be inserted. The SQLDA must contain a variable for each entry (number of rows * number of columns). The first row is placed in SQLDA variables 0 to (columns per row)-1, and so on.

**Note**
For scroll (value-sensitive) cursors, the inserted row will appear if the new row matches the WHERE clause and the keyset cursor has not finished populating. For dynamic cursors, if the inserted row matches the WHERE clause, the row may appear.Insensitive cursors cannot be updated.
Privileges
You must be the owner of the tables referenced in the cursor, or have INSERT privileges on the tables, or have the INSERT ANY TABLE system privilege.

Side effects
When inserting rows into a value-sensitive (keyset driven) cursor, the inserted rows appear at the end of the result set, even when they do not match the WHERE clause of the query or if an ORDER BY clause would normally have placed them at another location in the result set.

See also
● “UPDATE statement” on page 1037
● “UPDATE (positioned) statement [ESQL] [SP]” on page 1032
● “DELETE statement” on page 741
● “DELETE statement (positioned) [ESQL] [SP]” on page 739
● “INSERT statement” on page 860
● “SET statement” on page 985
● “Cursors used to modify rows” [SQL Anywhere Server - Programming]

Standards and compatibility
● SQL/2008 Vendor extension.

Example
The following statement illustrates the use of PUT in embedded SQL:

```
EXEC SQL PUT cur_employee FROM :employeeID, :surname;
```

RAISERROR statement
Signals an error and sends a message to the client.

Syntax
```
RAISERROR error-number [ format-string ] [, arg-list ]
```

Parameters
- `error-number` The error-number is a five-digit integer greater than 17000. The error number is stored in the global variable @@error.
- `format-string` format-string is string up to 255 bytes in length. It can contain placeholders for the arguments in the optional argument list. These placeholders are of the form %nn!, where nn is an integer between 1 and 20.

Remarks
The RAISERROR statement allows user-defined errors to be signaled and sends a message on the client.

To create new error messages, use the CREATE ERROR statement. To view the messages in the ISYSUSERMESSAGE system table, query the SYSUSERMESSAGE system view.
The extended values supported by the Adaptive Server Enterprise RAISERROR statement are not supported in SQL Anywhere.

If *format-string* is not supplied or is empty, the error number is used to locate an error message in the system tables. Adaptive Server Enterprise obtains messages 17000-19999 from the SYSMESSAGES table. In SQL Anywhere this table is an empty view, so errors in this range should provide a format string. Messages for error numbers of 20000 or greater are obtained from the ISYSUSERMESSAGE system table.

Intermediate RAISERROR status and code information is lost after the procedure terminates. If at return time an error occurs along with the RAISERROR then the error information is returned and the RAISERROR information is lost. The application can query intermediate RAISERROR statuses by examining @@error global variable at different execution points.

**Privileges**

None.

**Side effects**

None.

**See also**

- “BEGIN statement” on page 523
- “CREATE MESSAGE statement [T-SQL]” on page 614
- “SYSSMMESSAGE system view” on page 1414
- “CREATE TRIGGER statement [T-SQL]” on page 720
- “CREATE TRIGGER statement” on page 713
- “on_ts_1_error option” [SQL Anywhere Server - Database Administration]
- “login_procedure option” [SQL Anywhere Server - Database Administration]
- “continue_after_raiserror option” [SQL Anywhere Server - Database Administration]
- “SIGNAL statement [SP]” on page 992

**Standards and compatibility**

- **SQL/2008** Vendor extension.
- **Transact-SQL** The RAISERROR statement cannot be used in Transact-SQL compound statements and procedures.

**Example**

This example uses RAISERROR to disallow connections.

```sql
CREATE PROCEDURE DBA.login_check()
BEGIN
    // Allow a maximum of 3 concurrent connections
    IF( DB_PROPERTY('ConnCount') > 3 ) THEN
        RAISERROR 28000
            'User %1! is not allowed to connect -- there are ' ||
            'already %2! users logged on',
        Current User,
        CAST( DB_PROPERTY( 'ConnCount' ) AS INT )-1;
    ELSE
```

---

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 921
CALL sp_login_environment;
END IF;
END

GRANT EXECUTE ON DBA.login_check TO PUBLIC

SET OPTION PUBLIC.login_procedure='DBA.login_check'

READ statement [Interactive SQL]

Reads Interactive SQL statements from a file.

Syntax

```
READ [ ENCODING encoding ] filename [ parameter ] ...
```

Parameters

**ENCODING**  The ENCODING clause allows you to specify the encoding that is used to read the file. The READ statement does not process escape characters when it reads a file. It assumes that the entire file is in the specified encoding.

When running Interactive SQL, the encoding that is used to read the data is determined in the following order:

1. The encoding specified by the ENCODING clause (if this clause is specified).
2. The encoding specified by the byte order mark (BOM) in the file (if a BOM is specified).
3. The encoding specified with the default_isql_encoding option (if this option is set).
4. The default encoding for the platform you are running on. On English Windows computers, the default encoding is 1252.

Remarks

The READ statement reads a sequence of Interactive SQL statements from the named file. This file can contain any valid Interactive SQL statements, including other READ statements. READ statements can be nested to any depth.

If *filename* has no file extension, Interactive SQL searches for the same file name with the extension *.sql*.

If *filename* does not contain an absolute path, Interactive SQL searches for the file. The location of *filename* is determined based on the location of the READ statement, as follows:

- If the READ statement is executed directly in Interactive SQL, Interactive SQL first attempts to resolve the path to *filename* relative to the directory in which Interactive SQL is running. If unsuccessful, Interactive SQL looks for *filename* in the directories specified in the environment variable SQLPATH, and then the directories specified in the environment variable PATH.
If the READ statements reside in an external file (for example, a .sql file), Interactive SQL first attempts to resolve the path to filename relative to the location of the external file. If unsuccessful, Interactive SQL looks for filename in a path relative to the directory in which Interactive SQL is running. If still unsuccessful, Interactive SQL looks in the directories specified in the environment variable SQLPATH, and then the directories specified in the environment variable PATH.

Parameters can be listed after the name of the SQL script file. These parameters correspond to the parameters named in the PARAMETERS statement at the beginning of the statement file.

Parameter names must be enclosed in square brackets. Interactive SQL substitutes the corresponding parameter wherever the source file contains \{ parameter-name \}, where parameter-name is the name of the appropriate parameter.

The parameters passed to a script file can be identifiers, numbers, quoted identifiers, or strings. When quotes are used around a parameter, the quotes are put into the text during the substitution. Parameters that are not identifiers, numbers, or strings (contain spaces or tabs) must be enclosed in square brackets ([ ]). This allows for arbitrary textual substitution in the script file.

If not enough parameters are passed to the script file, Interactive SQL prompts for values for the missing parameters.

When executing a reload.sql file with Interactive SQL, you must specify the encryption key as a parameter. If you do not provide the key in the READ statement, Interactive SQL prompts for the key.

Privileges
None.

Side effects
None.

See also
- “PARAMETERS statement [Interactive SQL]” on page 912
- “default_isql_encoding option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “Interactive SQL” [SQL Anywhere Server - Database Administration]
- “Interactive SQL utility (dbisql)” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
This example reads from a fictitious files status.rpt and birthday.sql:

```
READ status.rpt '160';
READ birthday.sql [>= '1988-1-1'] [<= '1988-1-30'];
```

This example starts Interactive SQL and instructs it to process a fictitious file that uses a specific OEM codepage:

```
dsibql READ ENCODING 'cp437' myfile.sql
```
READTEXT statement [T-SQL]

Reads text and image values from the database, starting from a specified offset and reading a specified number of bytes. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

Syntax

```
READTEXT table-name.column-name
text-pointer-offset text-size
[ HOLDLOCK ]
```

Remarks

READTEXT is used to read CHAR, NCHAR, and BINARY columns from a database. You cannot perform READTEXT operations on views.

Privileges

You must be the owner or have SELECT privileges on the table, or have the SELECT ANY TABLE system privilege.

Side effects

None.

See also

- “WRITETEXT statement [T-SQL]” on page 1054
- “GET DATA statement [ESQL]” on page 822
- “TEXTPTR function [Text and image]” on page 385

Standards and compatibility

- SQL/2008   Transact-SQL extension.

Example

The following embedded SQL example uses TEXTPTR to locate the Description column associated with ProductID 500 in the MarketingInformation table.

The text pointer is stored in the variable `txtptr` and supplied as a parameter to the READTEXT statement which returns 55 bytes, starting at column offset 181.

```
EXEC SQL BEGIN DECLARE SECTION;
char            hostvar[100];
EXEC SQL END DECLARE SECTION;

EXEC SQL create variable txtptr binary(16);
EXEC SQL set txtptr =
  ( SELECT textptr(Description)
    FROM GROUPO.MarketingInformation
    WHERE ProductID = '500' );
EXEC SQL PREPARE S1 FROM
  'READTEXT GROUPO.MarketingInformation.Description txtptr 181 55';
EXEC SQL EXECUTE S1 INTO :hostvar;
printf( "hostvar: %s\n", hostvar );
```
REFRESH MATERIALIZED VIEW statement

Initializes or refreshes the data in a materialized view by executing its query definition.

Syntax

```
REFRESH MATERIALIZED VIEW view-list
  [ WITH {
    ISOLATION LEVEL isolation-level
    | { EXCLUSIVE | SHARE } MODE }
  [ FORCE BUILD ]

view-list : [ owner.]materialized-view-name [, ... ]

isolation-level :
  READ UNCOMMITTED
  | READ COMMITTED
  | SERIALIZABLE
  | REPEATABLE READ
  | SNAPSHOT
```

Parameters

**WITH clause**  Use the WITH clause to specify the type of locking to use on the underlying base tables during the refresh. The type of locking determines how the materialized view is populated and how concurrency for transactions is affected. The WITH clause setting does not impact the type of lock placed on the materialized view itself, which is always an exclusive lock. The possible locking clauses you can specify are:

- **ISOLATION LEVEL isolation-level**  Use WITH ISOLATION LEVEL to change the isolation level for the execution of the refresh operation. The original isolation level is restored for the connection when the statement completes.

  For immediate views, `isolation-level` can only be SERIALIZABLE.

  For snapshot isolation, only the transaction snapshot level is supported by the REFRESH MATERIALIZED VIEW statement. You should specify SNAPSHOT as the isolation level. The statement-snapshot and readonly-statement-snapshot levels are not supported.

- **EXCLUSIVE MODE**  Use WITH EXCLUSIVE MODE if you do not want to change the isolation level, but want to guarantee that the data is updated to be consistent with committed data in the underlying tables. When using WITH EXCLUSIVE MODE, exclusive table locks are placed on all underlying base tables and no other transaction can execute queries, updates, or any other action against the underlying table(s) until the refresh operation is complete. If exclusive table locks cannot be obtained, the refresh operation fails and an error is returned.

- **SHARE MODE**  Use WITH SHARE MODE to give read access on underlying tables to other transactions while the refresh operation takes place. When this clause is specified, shared table locks
are obtained on all underlying base tables before the refresh operation is performed and until the
refresh operation completes.

**FORCE BUILD clause** By default, when you execute a REFRESH MATERIALIZED VIEW
statement, the database server checks whether the materialized view is stale (that is, underlying tables
have changed since the materialized view was last refreshed). If it is not stale, the refresh does not take
place. Specify the FORCE BUILD clause to force a refresh of the materialized view regardless of whether
the materialized view is stale.

**Remarks**

Use this statement to initialize or refresh the materialized views listed in `view-list`.

If a REFRESH MATERIALIZED VIEW statement is executed against a materialized view that is not
stale, a refresh is not performed unless the FORCE BUILD clause is specified.

The default refresh behavior for locking and data concurrency is as follows:

- If the view is an immediate view, the default refresh behavior is WITH SHARE MODE, regardless of
  whether snapshot isolation is enabled.
- If the view is a manual view and snapshot isolation *is in use*, the default is WITH ISOLATION LEVEL
  SNAPSHOT.
- If the view is a manual view and snapshot isolation *is not in use*, the default is WITH SHARE MODE.

Several options need to have specific values for a REFRESH MATERIALIZED VIEW to succeed, and
for the view to be used in optimization. Additionally, there are option settings that are stored for each
materialized view when it is created. To refresh the view, or to use the view in optimization these option
settings must match the current options.

When a refresh fails after having done partial work, the view is left in an uninitialized state, and the data
cannot be restored to what it was before the refresh started. Examine the error that occurred when the
refresh failed, resolve the issue that caused the failure, and execute the REFRESH MATERIALIZED
VIEW statement again.

You can also use the IMMEDIATE REFRESH clause of the ALTER MATERIALIZED VIEW statement
to change the view to be refreshed immediately when underlying data changes.

This statement cannot be executed when the connection has cursors opened with the WITH HOLD clause
that use either statement or transaction snapshots.

**Privileges**

You must be the owner of the materialized view or have INSERT privilege on it. Additionally, you must
be the owner of the underlying tables, or have SELECT privilege on them, or have the SELECT ANY
TABLE system privilege.

**Side effects**

Any open cursors that reference the materialized view are closed.

A checkpoint is performed at the beginning of execution.
Automatic commits are performed at the beginning and end of execution. While executing, an exclusive schema lock is placed on the materialized view being refreshed using the connection blocking option, and shared schema locks, without blocking, are placed on all tables referenced by the materialized view. If the WITH clause is specified, extra locks may be acquired on the underlying tables. Also, until refreshing is complete, the materialized view is in an uninitialized state, making it unavailable to the database server for either query optimization or query execution.

If the REFRESH MATERIALIZED VIEW statement executes using snapshot isolation, the database's transaction log will contain both the REFRESH statement text and copies of all of the individual rows that are inserted to the materialized view. The individual rows are necessary to ensure that, should the database require recovery, the contents of the view after recovery matches precisely the view's contents upon the original completion of the REFRESH MATERIALIZED VIEW statement. Moreover, the individual rows in the transaction log are applied individually when the database is mirrored. For this reason, you may want to limit the frequency of REFRESH MATERIALIZED VIEW statements when using snapshot isolation, or truncate the transaction log periodically, using the BACKUP statement, to reduce the amount of disk space required for the transaction log.

See also
● “Materialized views” [SQL Anywhere Server - SQL Usage]
● “Materialized views restrictions” [SQL Anywhere Server - SQL Usage]
● “ALTER MATERIALIZED VIEW statement” on page 450
● “BACKUP statement” on page 516
● “CREATE MATERIALIZED VIEW statement” on page 612
● “sa_refresh_materialized_views system procedure” on page 1210
● “sa_materialized_view_info system procedure” on page 1176
● “sa_materialized_view_can_be_immediate system procedure” on page 1174

Standards and compatibility
● SQL/2008  Vendor extension.

Example
Suppose you create a materialized view, EmployeeConfid99, and then populate it with data using the following statements:

```sql
CREATE MATERIALIZED VIEW EmployeeConfid99 AS
    SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary, ManagerID,
            Departments.DepartmentName, Departments.DepartmentHeadID
    FROM GROUPO.Employees, GROUPO.Departments
    WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid99;
```

Later, after the view has been in use, you want to refresh the view using the READ COMMITTED isolation level (isolation level 1), and you want the view to be rebuilt. You could execute the following statement:

```sql
REFRESH MATERIALIZED VIEW EmployeeConfid99
    WITH ISOLATION LEVEL READ COMMITTED
    FORCE BUILD;
```
Caution
When you are done with this example, you should drop the materialized view you created (DROP MATERIALIZED VIEW EmployeeConfid99). Otherwise, you cannot make schema changes to its underlying tables, Employees and Departments, when trying out other examples. You cannot alter the schema of a table that has enabled, dependent materialized view. See “Dropping a materialized view” [SQL Anywhere Server - SQL Usage].

REFRESH TEXT INDEX statement

Refreshes a text index.

Syntax

```
REFRESH TEXT INDEX text-index-name ON [ owner.]table-name
[ WITH {
    ISOLATION LEVEL isolation-level
    | EXCLUSIVE MODE
    | SHARE MODE ]
[ FORCE { BUILD | INCREMENTAL } ]
```

Parameters

**WITH clause** Use the WITH clause to specify what kind of locks to obtain on the underlying base tables during the refresh. The types of locks obtained determine how the text index is populated and how concurrency for transactions is affected. If you do not specify the WITH clause, the default is WITH ISOLATION LEVEL READ UNCOMMITTED, regardless of any isolation level set for the connection.

You can specify the following WITH clause options:

- **ISOLATION LEVEL isolation-level** Use WITH ISOLATION LEVEL to change the isolation level for the execution of the refresh operation.
  
  The original isolation level of the connection is restored at the end of the statement execution.

- **EXCLUSIVE MODE** Use WITH EXCLUSIVE MODE if you do not want to change the isolation level, but want to guarantee that the data is updated to be consistent with committed data in the underlying table. When using WITH EXCLUSIVE MODE, exclusive table locks are placed on the underlying base table and no other transaction can execute queries, updates, or any other action against the underlying table(s) until the refresh operation is complete. If table locks cannot be obtained, the refresh operation fails and an error is returned.

- **SHARE MODE** Use WITH SHARE MODE to give read access on the underlying table to other transactions while the refresh operation takes place. When this clause is specified, shared table locks are obtained on the underlying base table before the refresh operation is performed and are held until the refresh operation completes.

**FORCE clause** Use this clause to specify the refresh method. If this clause is not specified, the database server decides whether to do an incremental update or a full rebuild based on how much of the table has changed.
FORCE BUILD clause Refreshes the text index by recreating it. Use this clause to force a complete rebuild of the text index.

FORCE INCREMENTAL clause Refreshes the text index based only on what has changed in the underlying table. An incremental refresh takes less time to complete if there have not been a significant amount of updates to the underlying table. Use this clause to force an incremental update of the text index.

An incremental refresh does not remove deleted entries from the text index. As a result, the size of the text index may be larger than expected to contain the current and historic data. Typically, this issue occurs with text indexes that are always manually refreshed with the FORCE INCREMENTAL clause. On automatically refreshed text indexes, historic data is automatically deleted when it makes up 50% of the total size of the text index.

Remarks
This statement can only be used on text indexes defined as MANUAL REFRESH or AUTO REFRESH.

When using the FORCE clause, you can examine the results of the sa_text_index_stats system procedure to decide whether a complete rebuild (FORCE BUILD), or incremental update (FORCE INCREMENTAL) is most appropriate.

You cannot execute the REFRESH TEXT INDEX statement on a text index that is defined as IMMEDIATE REFRESH.

For MANUAL REFRESH text indexes, use the sa_text_index_stats system procedure to determine whether the text index should be refreshed. Divide pending_length by doc_length, and use the percentage as a guide for deciding whether a refresh is required. To determine the type of rebuild required, use the same process for deleted_length and doc_count.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.

Privileges
You must be the owner of the table, or have one of the following privileges:

- REFERENCES privilege on the table
- CREATE ANY INDEX system privilege
- ALTER ANY INDEX system privilege
- CREATE ANY OBJECT system privilege
- ALTER ANY OBJECT system privilege

Side effects
Automatic commit.
See also

- “Text index concepts and reference” [SQL Anywhere Server - SQL Usage]
- “Transactions and isolation levels” [SQL Anywhere Server - SQL Usage]
- “Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]
- “CREATE TEXT INDEX statement” on page 711
- “ALTER TEXT INDEX statement” on page 504
- “DROP TEXT INDEX statement” on page 782
- “TRUNCATE TEXT INDEX statement” on page 1021
- “Table locks” [SQL Anywhere Server - SQL Usage]
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]
- “Text index refresh types” [SQL Anywhere Server - SQL Usage]
- “sa_refresh_text_indexes system procedure” on page 1211
- “sa_text_index_stats system procedure” on page 1256

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement refreshes a fictitious text index called MarketingTextIndex, forcing it to be rebuilt.

```
REFRESH TEXT INDEX MarketingTextIndex ON GROUPO.MarketingInformation
FORCE BUILD;
```

**REFRESH TRACING LEVEL statement**

Reloads the tracing levels from the sa_diagnostic_tracing_level table while a tracing session is in progress.

Syntax

```
REFRESH TRACING LEVEL
```

Remarks

This statement is used to reload the tracing level information from the sa_diagnostic_tracing_level table. It must be called from a database that is being profiled.

When a tracing session is first started, rows from the sa_diagnostic_tracing_level table are loaded into server memory to control what kind of information is traced. To change the types of data being traced without stopping and restarting the tracing session, manually delete or insert the appropriate rows in the sa_diagnostic_tracing_level table, and then execute the REFRESH TRACING LEVEL statement to reload the settings.

To see the current tracing levels, query the sa_diagnostic_tracing_level table as follows:

```
SELECT * FROM sa_diagnostic_tracing_level WHERE enabled = 1;
```

Suppose you are troubleshooting a performance problem. You turn on a high level of tracing for the entire database to capture the queries that are causing the problem. After starting the tracing session, you find
that capturing all queries for all users on your system slows down your database too much, so you decide you would rather limit tracing to one user and wait for that user to report a problem. However, you do not want to stop the tracing session to change the settings.

You can do this in Sybase Central by using the Database Tracing wizard, which is the recommended method. However, you can also do this from the command line by replacing the rows in `sa_diagnostic_tracing_level` table where `scope=DATABASE` and `enabled=1`, with equivalent rows where `scope=USER`, `identifier=userid`, `enabled=1`, and so on. Then, you execute a `REFRESH TRACING LEVEL` statement to continue tracing using the new settings.

**Privileges**

You must have the MANAGE PROFILING system privilege.

**Side effects**

None.

**See also**

- “ATTACH TRACING statement” on page 514
- “DETACH TRACING statement” on page 750
- “Diagnostic tracing” [SQL Anywhere Server - SQL Usage]
- “sa_diagnostic_tracing_level table” on page 1081

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

This example refreshes the tracing level:

```sql
REFRESH TRACING LEVEL;
```

**RELEASE SAVEPOINT statement**

Releases a savepoint within the current transaction.

**Syntax**

```sql
RELEASE SAVEPOINT [ savepoint-name ]
```

**Remarks**

Release a savepoint. The `savepoint-name` is an identifier specified on a SAVEPOINT statement within the current transaction. If `savepoint-name` is omitted, the most recent savepoint is released.

Releasing a savepoint does not do any type of COMMIT. It simply removes the savepoint from the list of currently active savepoints.

There must have been a corresponding SAVEPOINT within the current transaction.
Privileges

None.

Side effects

None.

See also

- “BEGIN TRANSACTION statement [T-SQL]” on page 527
- “COMMIT statement” on page 541
- “ROLLBACK statement” on page 950
- “ROLLBACK TO SAVEPOINT statement” on page 951
- “SAVEPOINT statement” on page 954
- “Savepoints within transactions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 RELEASE SAVEPOINT is part of optional SQL/2008 language feature T271, "Savepoints".

REMOTE RESET statement [SQL Remote]

Starts all subscriptions for a remote user in a single transaction in custom database-extraction procedures.

Syntax

REMOTE RESET userid

Remarks

This statement starts all subscriptions for a remote user in a single transaction. It sets the log_sent and confirm_sent values in ISYSGREMOTEUSER table to the current position in the transaction log. It also sets the created and started values in ISYSSUBSCRIPTION to the current position in the transaction log for all subscriptions for this remote user. The statement does not do a commit. You must do an explicit commit after this call.

To write an extraction process that is safe on a live database, the data must be extracted at isolation level 3 in the same transaction as the subscriptions are started.

This statement is an alternative to start subscription. START SUBSCRIPTION has an implicit commit as a side effect, so that if a remote user has several subscriptions, it is impossible to start them all in one transaction using START SUBSCRIPTION.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

No automatic commit is done by this statement.
See also
- “START SUBSCRIPTION statement [SQL Remote]” on page 1000
- “SYSDATABASE system view” on page 860
- “Creating multiple remote databases” [SQL Remote]
- “Starting subscriptions” [SQL Remote]

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement resets the subscriptions for remote user Sam_Singer:

```
REMOTE RESET Sam_Singer;
COMMIT;
```

### REMOVE EXTERNAL OBJECT statement

Removes an external object from the database.

**Syntax**
```
REMOVE EXTERNAL OBJECT object-name
```

**Parameters**
- `object-name` The name of the external object.

**Remarks**
For more information about external environments, see “SQL Anywhere external environment support” [SQL Anywhere Server - Programming].

**Privileges**
You must be the owner of the external object, or have the MANAGE ANY EXTERNAL OBJECT system privilege.

**Side effects**
None

See also
- “SQL Anywhere external environment support” [SQL Anywhere Server - Programming]
- “ALTER EXTERNAL ENVIRONMENT statement” on page 438
- “INSTALL EXTERNAL OBJECT statement” on page 866
- “START EXTERNAL ENVIRONMENT statement” on page 997
- “STOP EXTERNAL ENVIRONMENT statement” on page 1005
- “SYSEXTERNENV system view” on page 1358
- “SYSEXTERNENVOBJECT system view” on page 1360
Standards and compatibility

- SQL/2008  Vendor extension.

**REMOVE JAVA statement**

Removes a class or JAR file from a database.

**Syntax**

```sql
REMOVE JAVA

{ CLASS java-class-name [ , java-class-name ... ]
  | JAR jar-name [ , jar-name ... ] }
```

**Parameters**

- **CLASS clause**  The `java-class-name` parameter is the name of one or more Java class to be removed. These classes must be installed classes in the current database.

- **JAR clause**  The `jar-name` is a single-quoted character string value of maximum length 255. Each `jar-name` must be equal to the `jar-name` of a retained JAR in the current database. Equality of `jar-name` is determined by the character string comparison rules of the SQL system.

**Remarks**

Removes a class or JAR file from the database. The class or JAR must already be installed.

This statement is not supported on Windows Mobile.

**Privileges**

You must be the owner of the class or JAR file, or have the MANAGE ANY EXTERNAL OBJECT system privilege.

**See also**

- “INSTALL JAVA statement” on page 868
- “SYSJAR system view” on page 1367
- “SYSJARCOMPONENT system view” on page 1367
- “SYSJAVACLASS system view” on page 1368

**Standards and compatibility**

- SQL/2008  Vendor extension.

**Example**

This example removes a fictitious Java class named Demo from the current database:

```sql
REMOVE JAVA CLASS Demo;
```

This example removes a fictitious Java JAR named myJar from the current database:

```sql
REMOVE JAVA JAR 'myJar';
```
REORGANIZE TABLE statement

Defragments tables when a full rebuild of the database is not possible due to the requirements for continuous access to the database.

Syntax

```
REORGANIZE TABLE [ owner.]table-name
  [ { PRIMARY KEY
     | FOREIGN KEY foreign-key-name
     | INDEX index-name } ]
```

Parameters

Reorganize the table according to the values in one of the following:

- **PRIMARY KEY clause**  Reorganizes the primary key index for the table.
- **FOREIGN KEY clause**  Reorganizes the specified foreign key.
- **INDEX clause**  Reorganizes the specified index.

Remarks

Table fragmentation can impede performance. Use this statement to defragment rows in a table, or to compress indexes which have become sparse due to DELETEs. It may also reduce the total number of pages used to store the table and its indexes, and it may reduce the number of levels in an index tree. However, it will not result in a reduction of the total size of the database file. It is recommended that you use the sa_table_fragmentation and sa_index_density system procedures to select tables worth processing.

If an index or key is not specified, the reorganization process defragments rows in the table by deleting and re-inserting groups of rows. For each group, an exclusive lock on the table is obtained. Once the group has been processed, the lock is released and re-acquired (waiting if necessary), providing an opportunity for other connections to access the table. Checkpoints are suspended while a group is being processed; once a group is finished, a checkpoint may occur. The rows are processed in order of the clustered index if one exists; otherwise, they are processed in order of the primary key. If the table does not have a clustered index or a primary key, an error is returned. The processed rows are re-inserted at the end of the table, resulting in the rows being clustered by primary key at the end of the process. The same amount of work is required, regardless of how fragmented the rows initially were.

If an index or key is specified, the specified index is processed. For the duration of the operation, an exclusive lock is held on the table and checkpoints are suspended. Any attempts to access the table by other connections will block or fail, depending on their setting of the blocking option. The duration of the lock is minimized by pre-reading the index pages before obtaining the exclusive lock.

Since reorganization may modify many pages, the checkpoint log can become large. This can result in an increase in the database file size. However, this increase is temporary since the checkpoint log is deleted at shutdown and the file is truncated at that point.

This statement is not logged to the transaction log.

This statement cannot be executed when there are cursors opened with the WITH HOLD clause that use either statement or transaction snapshots.
During the execution of this statement, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

This statement is not supported on Windows Mobile.

**Privileges**

You must be the owner of the table, or have the REORGANIZE ANY OBJECT system privilege.

**Side effects**

Before starting the reorganization, a checkpoint is done to try to maximize the number of free pages. Also, when executing the REORGANIZE TABLE statement, there is an implied commit for approximately every 100 rows, so reorganizing a large table causes multiple commits to take place.

**Standards and compatibility**

- **SQL/2008** Vendor extension.

**See also**

- “Progress connection property” [SQL Anywhere Server - Database Administration]
- “progress_messages option” [SQL Anywhere Server - Database Administration]
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]

**Examples**

The following statement reorganizes the primary key index for the Employees table:

```sql
REORGANIZE TABLE GROUPO.Employees
PRIMARY KEY;
```

The following statement reorganizes the table pages of the Employees table:

```sql
REORGANIZE TABLE GROUPO.Employees;
```

The following statement reorganizes the index IX_product_name on the Products table:

```sql
REORGANIZE TABLE GROUPO.Products
INDEX IX_product_name;
```

The following statement reorganizes the foreign key FK_DepartmentID_DepartmentID for the Employees table:

```sql
REORGANIZE TABLE GROUPO.Employees
FOREIGN KEY FK_DepartmentID_DepartmentID;
```

**RESIGNAL statement [SP]**

Resignals an exception condition.

**Syntax**

```sql
RESIGNAL [ exception-name ]
```
Remarks
Within an exception handler, RESIGNAL allows you to quit the compound statement with the exception still active, or to quit reporting another named exception. The exception is handled by another exception handler or returned to the application.

Privileges
None.

Side effects
None.

See also
- “SIGNAL statement [SP]” on page 992
- “BEGIN statement” on page 523
- “Exception handlers” [SQL Anywhere Server - SQL Usage]
- “RAISERROR statement” on page 920

Standards and compatibility
- SQL/2008 The RESIGNAL statement is part of optional SQL/2008 language feature P002, "Computational completeness".
- Transact-SQL The RESIGNAL statement cannot be used in Transact-SQL compound statements and procedures.

Example
The example fragment returns all exceptions except SQLSTATE 52003 to the application.

```sql
... DECLARE COLUMN_NOT_FOUND EXCEPTION
    FOR SQLSTATE '52003';
... EXCEPTION
    WHEN COLUMN_NOT_FOUND THEN
    SET message='Column not found';
    WHEN OTHERS THEN
    RESIGNAL;
```

RESTORE DATABASE statement
Restores a backed up database from an archive.

Syntax
```
RESTORE DATABASE filename
    FROM archive-root
    [ CATALOG ONLY
        [ [ RENAME dbspace-name TO new-dbspace-name ] ... ]
    [ HISTORY { ON | OFF } ]
    [ KEY encryption-key ]
```
filename : string | variable
archive-root : string | variable
new-dbspace-name : string | variable

Parameters

FROM clause If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

CATALOG ONLY clause Retrieves information about the named archive, and places it in the backup history file (backup.syb), but does not restore any data from the archive.

RENAME clause Allows you to specify a new location for each dbspace. You cannot use the RENAME clause to change the dbspace name. However, you can use the RENAME clause to change the file name.

HISTORY clause Allows you to control whether the RESTORE DATABASE operation is recorded in the history file, backup.syb.

KEY clause Allows you to specify the encryption key to restore an archived strongly encrypted database that was backed up with free page elimination on. If the back up was made with free page elimination off, then you do not need to specify the encryption key to restore the database. The key can be either a string or a variable name.

As of version 12, you cannot restore archive backups created with version 11 or earlier database servers.

Remarks

Unless HISTORY OFF is specified, each RESTORE DATABASE operation updates a backup history file called backup.syb. This file records the BACKUP and RESTORE operations that have been performed on a database server. You may want to prevent the RESTORE DATABASE operation from being recorded in backup.syb if the following conditions apply:

● your RESTORE DATABASE operations occur frequently
● there is no procedure to periodically archive or delete the backup.syb file
● disk space is very limited

Note
If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

RESTORE DATABASE replaces the database that is being restored. If you need incremental backups, use the image format of the BACKUP command and save only the transaction log; however, image backups to tape are not supported.

During the execution of this statement, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.
You cannot be connected to the database you are restoring. You must be connected to a different
database. For example, connect to the utility database. The database that you are encrypting must not be
running.

This statement is not supported on Windows Mobile.

Privileges
Your ability to execute this statement depends on the setting for the -gu database option, and whether you
have the SERVER OPERATOR system privilege.

Side effects
None.

See also
- “Predefined dbspaces” [SQL Anywhere Server - Database Administration]
- “BACKUP statement” on page 516
- “Backup and data recovery” [SQL Anywhere Server - Database Administration]
- “SALOGDIR environment variable” [SQL Anywhere Server - Database Administration]
- “FREE PAGE ELIMINATION clause, BACKUP statement” on page 520
- “-gu database server option” [SQL Anywhere Server - Database Administration]
- “Progress connection property” [SQL Anywhere Server - Database Administration]
- “progress_messages option” [SQL Anywhere Server - Database Administration]

Standards and compatibility
- SQL/2008 Vendor extension.
- Windows Mobile Not supported on Windows Mobile.

Example
The following example restores a fictitious database from a tape drive. The number of backslashes that
are required depends on which database you are connected to when you execute RESTORE DATABASE.
The database affects the setting of the escape_character option. It is normally set to On, but is set to Off in
utility_db. When connected to any database other than utility_db, the extra backslashes are required.

```
RESTORE DATABASE 'd:\\dbhome\\mydatabase.db'
FROM '\\\\\\\\\tape0';
```

RESUME statement
Resumes execution of a cursor that returns result sets.

Syntax
```
RESUME cursor-name
```

cursor-name : identifier | hostvar
Remarks
This statement resumes execution of a procedure that returns result sets. The procedure executes until the next result set (SELECT statement with no INTO clause) is encountered. If the procedure completes and no result set is found, the SQLSTATE_PROCEDURE_COMPLETE warning is set. This warning is also set when you RESUME a cursor for a SELECT statement.

The RESUME statement is not supported in Interactive SQL. To view multiple result sets in Interactive SQL, set the isql_show_multiple_result_sets option to On, or choose Tools » Options, and then select Show Multiple Result Sets on the Results tab.

The cursor must have been previously opened.

Privileges
None.

Side effects
None.

See also
● “DECLARE CURSOR statement [ESQL] [SP]” on page 729
● “FETCH statement [ESQL] [SP]” on page 800
● “Result sets” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
● SQL/2008 Vendor extension.

Example
Following are embedded SQL examples.

```
EXEC SQL RESUME cur_employee;
EXEC SQL RESUME :cursor_var;
```

**RETURN statement**
Exits from a function, procedure, or batch unconditionally, optionally providing a return value.

Syntax

```
RETURN [ expression ]
```

Remarks
A RETURN statement causes an immediate exit from a block of SQL. If expression is supplied, the value of expression is returned as the value of the function or procedure. Subqueries cannot be used in expression.

If the RETURN appears inside an inner BEGIN block, it is the outer BEGIN block that is terminated.

Statements following a RETURN statement are not executed.
Within a function, the expression should be of the same data type as the function's RETURNS data type.

Within a procedure, RETURN is used for Transact-SQL compatibility, and is used to return an integer error code.

Privileges
None.

Side effects
None.

See also
- “CREATE FUNCTION statement” on page 594
- “CREATE FUNCTION statement [External call]” on page 580
- “CREATE FUNCTION statement [Web service]” on page 586
- “CREATE PROCEDURE statement” on page 639
- “CREATE PROCEDURE statement [External call]” on page 620
- “CREATE PROCEDURE statement [Web service]” on page 628
- “BEGIN statement” on page 523
- “Returning a value using the RETURN statement” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008 Core feature.

Example
The following function returns the product of three numbers:

```sql
CREATE FUNCTION product (  
  a NUMERIC,  
  b NUMERIC,  
  c NUMERIC )  
RETURNS NUMERIC  
BEGIN  
  RETURN ( a * b * c );  
END;
```

Calculate the product of three numbers:

```sql
SELECT product(2, 3, 4);
```

<table>
<thead>
<tr>
<th>product(2, 3, 4)</th>
<th>24.000000</th>
</tr>
</thead>
</table>

The following procedure uses the RETURN statement to avoid executing a complex query if it is meaningless:

```sql
CREATE PROCEDURE customer_products  
( in customer_ID integer DEFAULT NULL)  
RESULT ( ID integer, quantity_ordered integer )  
BEGIN
```
IF customer_ID NOT IN (SELECT ID FROM Customers)
OR customer_ID IS NULL THEN
RETURN
ELSE
SELECT Products.ID,sum(SalesOrderItems.Quantity)
FROM GROUPO.Products,
SalesOrderItems,
SalesOrders
WHERE SalesOrders.CustomerID=customer_ID
AND SalesOrders.ID=SalesOrderItems.ID
AND SalesOrderItems.ProductID=Products.ID
GROUP BY Products.ID
END IF
END;

REVOKE CONSOLIDATE statement [SQL Remote]

Stops a consolidated database from receiving SQL Remote messages from this database.

Syntax
REVOKE CONSOLIDATE FROM userid

Remarks
CONSOLIDATE privileges must be granted at a remote database for the user ID representing the consolidated database. The REVOKE CONSOLIDATE statement removes the consolidated database user ID from the list of users receiving messages from the current database.

Privileges
You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects
Automatic commit. Drops all subscriptions for the user.

See also
- “GRANT CONSOLIDATE statement [SQL Remote]” on page 834
- “GRANT ROLE SYS_REPLICATION_ADMIN_ROLE statement [MobiLink] [SQL Remote]” on page 840
- “REVOKE PUBLISH statement [SQL Remote]” on page 943
- “REVOKE REMOTE statement [SQL Remote]” on page 944

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following statement revokes consolidated status from the Sam_Singer user ID:

REVOKE CONSOLIDATE FROM Sam_Singer;
REVOKE PUBLISH statement [SQL Remote]

Terminates the identification of the named user ID as the current publisher. You must have the SYS_REPLICATION_ADMIN_ROLE system role to revoke publisher rights.

Syntax

REVOKE PUBLISH FROM userid

Remarks

Each database in a SQL Remote installation is identified in outgoing messages by a publisher user ID. The current publisher user ID can be determined by querying the CURRENT PUBLISHER special value as follows:

```
SELECT CURRENT PUBLISHER;
```

The REVOKE PUBLISH statement ends the identification of the named user ID as the publisher. To change publishers, you must REVOKE PUBLISH from the current publisher, and then GRANT PUBLISH to the new publisher.

If you change the publisher user ID at any consolidated or remote database in a SQL Remote installation, you must ensure that the new publisher user ID is granted REMOTE privilege on all databases receiving messages from the database. Making this change requires all subscriptions to be dropped and recreated.

You should not revoke the publisher while the database has active SQL Remote publications or subscriptions.

Revoking publisher and not granting it to a new user has consequences for a SQL Remote installation:

- You cannot insert data into any tables with a CURRENT PUBLISHER column as part of the primary key. Any outgoing messages are not identified with a publisher user ID, and so are not accepted by recipient databases.

Executing this statement changes the value of the PUBLIC.db_publisher database option.

Privileges

You must have the SET ANY SYSTEM OPTION system privilege.

Side effects

Automatic commit.

See also

- “GRANT ROLE SYS_REPLICATION_ADMIN_ROLE statement [MobiLink] [SQL Remote]” on page 840
- “db_publisher option” [SQL Anywhere Server - Database Administration]
- “GRANT PUBLISH statement [SQL Remote]” on page 839
- “REVOKE REMOTE statement [SQL Remote]” on page 944
- “REVOKE CONSOLIDATE statement [SQL Remote]” on page 942
- “CURRENT PUBLISHER special value” on page 66
Standards and compatibility

- SQL/2008 Vendor extension.

Example

Terminate the identification of publisher_ID as the current publisher.

```
REVOKE PUBLISH FROM publisher_ID;
```

**REVOKE REMOTE statement [SQL Remote]**

Stops a user from being able to receive SQL Remote messages from this database.

**Syntax**

```
REVOKE REMOTE FROM userid, ...
```

**Remarks**

REMOTE privilege is required for a user ID to receive messages in a SQL Remote replication installation. The REVOKE REMOTE statement removes a user ID from the list of users receiving messages from the current database.

**Privileges**

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

**Side effects**

Automatic commit. Drops all subscriptions for the user.

**See also**

- “REVOKE PUBLISH statement [SQL Remote]” on page 943
- “GRANT REMOTE statement [SQL Remote]” on page 844
- “REVOKE ROLE SYS_REPLICATION_ADMIN_ROLE statement [MobiLink] [SQL Remote]” on page 948
- “REVOKE CONSOLIDATE statement [SQL Remote]” on page 942
- “Revoking REMOTE privilege” [SQL Remote]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

```
REVOKE REMOTE FROM Sam_Singer;
```

**REVOKE statement**

Revolves roles and privileges from users and roles.
### Revoke system roles

```
REVOKE ROLE system-role
FROM grantee, ...
```

- `grantee`:
  - `{ system-role | userid }
- `system-role`:
  - `dbo`
  - `DIAGNOSTICS`
  - `PUBLIC`
  - `rs_systabgroup`
  - `SA_DEBUG`
  - `SYS`
  - `SYS_REPLICATION_ADMIN_ROLE`
  - `SYS_RUN_REPLICATION_ROLE`
  - `SYS_SAMONITOR_ROLE`
  - `SYS_SPATIAL_ADMIN_ROLE`

### Revoke user-defined roles

```
REVOKE [ ADMIN OPTION FOR ] ROLE user-defined-role
FROM grantee, ...
```

- `grantee`:
  - `{ system-role | userid }

### Revoke compatibility roles

```
REVOKE [ ADMIN OPTION FOR ] ROLE compatibility-role-name
FROM grantee, ...
```

- `compatibility-role-name`:
  - `SYS_AUTH_BACKUP_ROLE`
  - `SYS_AUTH_DBA_ROLE`
  - `SYS_AUTH_PROFILE_ROLE`
  - `SYS_AUTH_READCLIENTFILE_ROLE`
  - `SYS_AUTH_READFILE_ROLE`
  - `SYS_AUTH_RESOURCE_ROLE`
  - `SYS_AUTH_SA_ROLE`
  - `SYS_AUTH_SSO_ROLE`
  - `SYS_AUTH_VALIDATE_ROLE`
  - `SYS_AUTH_WRITECLIENTFILE_ROLE`
  - `SYS_AUTH_WRITEFILE_ROLE`

- `grantee`:
  - `{ system-role | userid }`
Revoke system privileges

REVOKE [ ADMIN OPTION FOR ] privilege
FROM grantee, ...

grantee :
{ system-role | userid }

Revoke object-level privileges

REVOKE object-level-privilege[,...]
ON [ owner.]table-or-view
FROM userid[,...]

object-level-privilege :
ALL [ PRIVILEGES ]
| ALTER
| DELETE
| INSERT
| LOAD
| REFERENCES [ ( column-name[,...] ) ]
| SELECT [ ( column-name[,...] ) ]
| TRUNCATE
| UPDATE [ ( column-name[,...] ) ]

Revoke CONNECT, INTEGRATED LOGIN, and KERBEROS LOGIN

REVOKE capability FROM userid[,...]

capability :
CONNECT
| INTEGRATED LOGIN
| KERBEROS LOGIN

Revoke EXECUTE on a procedure

REVOKE EXECUTE
ON [ owner.]procedure-name[,...]
FROM userid[,...]

Revoke USAGE on a sequence

REVOKE USAGE ON SEQUENCE sequence-name[,...]
FROM userid[,...]

Parameters

- **ADMIN OPTION FOR clause** Specify the ADMIN OPTION FOR clause to revoke administrative privileges. This clause revokes only the administrative privileges, while leaving the role granted.

- **REVOKE CONNECT** REVOKE CONNECT removes a user ID from a database, and also destroys any objects (tables, views, procedures, and so on) owned by that user. System privileges granted by the user remain in effect; however, object-level privileges granted by the user are revoked.

You cannot execute a REVOKE CONNECT statement on a user if the user being dropped owns a table referenced by a view owned by another user.
When you are connected to the utility database, executing `REVOKE CONNECT FROM DBA` disables future connections to the utility database. No future connections can be made to the utility database unless you use a connection that existed before the `REVOKE CONNECT` was executed, or restart the database server.

- **REVOKE USAGE ON SEQUENCE** Specify this syntax to remove the privilege to evaluate the current or next value in a sequence.

**Remarks**

If a role or privilege that is being revoked was not granted to `grantee`, then the statement does nothing, and does not return an error.

`REVOKE ROLE` fails with an error if, as a consequence of executing the statement, the number of administrators for the role or privileges being revoked would fall below the required minimum as set by the `min_role_admins` database option.

When you revoke object-level privileges from a role, grantees of that role inherit those changes.

When you revoke an object-level privilege for a user who also had administrative rights for that privilege, then everyone who that user granted the privilege to also has their privilege revoked, as well as anyone that the grantees granted it to, and so on.

If you are revoking connection-related privileges from a user, the user must not be connected to the database.

**Privileges**

You must have administration rights for the system privileges or roles that you are revoking.

If you are revoking object-level privileges, you must have one of the following:

- Ownership of the object
- Administrative rights on the object-level privilege for that object
- `MANAGE ANY OBJECT PRIVILEGE` system privilege

**Side effects**

Automatic commit.

**See also**

- “Viewing the roles and privileges for a user or role (Sybase Central)” [SQL Anywhere Server - Database Administration]
- “System privileges” [SQL Anywhere Server - Database Administration]
- “Object-level privileges” [SQL Anywhere Server - Database Administration]
- “Roles” [SQL Anywhere Server - Database Administration]
- “Authorities become compatibility roles” [SQL Anywhere Server - Database Administration]
- “`min_role_admins` option” [SQL Anywhere Server - Database Administration]
- “`GRANT` statement” on page 827
Standards and compatibility

- **SQL/2008** REVOKE capability is a vendor extension. REVOKE object-level-privilege and REVOKE EXECUTE are core features of the SQL/2008 standard. With REVOKE ALL (revoking all object-level privileges), the PRIVILEGES keyword is optional in SQL Anywhere, while in the SQL/2008 standard it is mandatory.

REVOKE USAGE ON SEQUENCE is part of optional SQL/2008 language feature T176, "Sequence generator support".

Example

This example prevents user Dave from updating the Employees table.

```sql
REVOKE UPDATE ON GROUPO.Employees FROM Dave;
```

This example revokes the SYS_AUTH_VALIDATE_ROLE compatibility role from fictitious user Jim.

```sql
REVOKE ROLE SYS_AUTH_VALIDATE_ROLE FROM Jim;
```

This example revokes the DIAGNOSTICS system role from a fictitious user named Administrator.

```sql
REVOKE ROLE DIAGNOSTICS FROM Administrator;
```

This example prevents a fictitious user-extended role called Finance from executing the procedure ShowCustomers.

```sql
REVOKE EXECUTE ON ShowCustomers FROM Finance;
```

This example drops user FranW from the database. This syntax is deprecated; consider using the DROP USER statement instead.

```sql
REVOKE CONNECT FROM FranW;
```

This example revokes database login privilege from a fictitious Kerberos user, pchin.

```sql
REVOKE KERBEROS LOGIN FROM "pchin@MYREALM.COM";
```

**REVOKE ROLE SYS_REPLICATION_ADMIN_ROLE statement [MobiLink] [SQL Remote]**

Revokes the SYS_REPLICATION_ADMIN_ROLE system role from a user (grantee). Revoking this role removes the ability for the user to perform administrative tasks related to replication, such as granting replication roles, managing publications, subscriptions, synchronization users and profiles, managing message types, and setting replication-related options.

**Syntax**

```sql
REVOKE ROLE SYS_REPLICATION_ADMIN_ROLE
FROM grantee[, ...]
```

**Privileges**

You must have the SYS_REPLICATION_ADMIN_ROLE system role.
Side effects

Automatic commit.

See also

- “Roles” [SQL Anywhere Server - Database Administration]
- “GRANT ROLE SYS_REPLICATION_ADMIN_ROLE statement [MobiLink] [SQL Remote]” on page 840
- “REVOKE PUBLISH statement [SQL Remote]” on page 943
- “REVOKE REMOTE statement [SQL Remote]” on page 944
- “REVOKE CONSOLIDATE statement [SQL Remote]” on page 942
- “Synchronization initiation” [MobiLink - Client Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Examples

The following statement revokes SYS_REPLICATION_ADMIN_ROLE from user Sam_Singer.

```
REVOKE ROLE SYS_REPLICATION_ADMIN_ROLE FROM Sam_Singer;
```

REVOKE ROLE SYS_RUN_REPLICATION_ROLE statement
[MobiLink] [SQL Remote]

Revokes the SYS_RUN_REPLICATION_ROLE system role from a user (grantee). Revoking this role removes the ability for the user to:

- Run the dbremote utility for replication
- Run the dbmlsync utility for synchronization

Syntax

```
REVOKE ROLE SYS_RUN_REPLICATION_ROLE
FROM grantee[, ...]
```

Remarks

In MobiLink, SYS_RUN_REPLICATION_ROLE is a set of privileges required by the SQL Anywhere synchronization client (dbmlsync).

In SQL Remote, SYS_RUN_REPLICATION_ROLE enables the SQL Remote Message Agent to have full access to the database to make any changes contained in the messages.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.
See also

- “Roles” [SQL Anywhere Server - Database Administration]
- “GRANT ROLE SYS_RUN_REPLICATION_ROLE statement [MobiLink] [SQL Remote]” on page 842
- “REVOKE PUBLISH statement [SQL Remote]” on page 943
- “REVOKE REMOTE statement [SQL Remote]” on page 944
- “REVOKE CONSOLIDATE statement [SQL Remote]” on page 942
- “Synchronization initiation” [MobiLink - Client Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Examples

The following statement revokes SYS_RUN_REPLICATION_ROLE from user Sam_Singer.

```
REVOKE ROLE SYS_RUN_REPLICATION_ROLE FROM Sam_Singer;
```

**ROLLBACK statement**

Ends a transaction and undo any changes made since the last COMMIT or ROLLBACK.

Syntax

```
ROLLBACK [ WORK ]
```

Remarks

A transaction is the logical unit of work done on one database connection to a database between COMMIT or ROLLBACK statements. The ROLLBACK statement ends the current transaction and undoes all changes made to the database since the previous COMMIT or ROLLBACK.

Privileges

None.

Side effects

Closes all cursors not opened WITH HOLD.

See also

- “COMMIT statement” on page 541
- “Keyboard shortcuts for Interactive SQL” [SQL Anywhere Server - Database Administration]
- “ROLLBACK TO SAVEPOINT statement” on page 951
- “Interactive SQL options” [SQL Anywhere Server - Database Administration]
- “Cancellation of changes” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008 Core feature.
ROLLBACK TO SAVEPOINT statement
Cancels any changes made since a SAVEPOINT.

Syntax
ROLLBACK TO SAVEPOINT [ savepoint-name ]

Remarks
The ROLLBACK TO SAVEPOINT statement will undo any changes that have been made since the SAVEPOINT was established. Changes made before the SAVEPOINT are not undone; they are still pending.

The savepoint-name is an identifier that was specified on a SAVEPOINT statement within the current transaction. If savepoint-name is omitted, the most recent savepoint is used. Any savepoints since the named savepoint are automatically released.

There must have been a corresponding SAVEPOINT within the current transaction.

Privileges
None.

Side effects
None.

See also
● “BEGIN TRANSACTION statement [T-SQL]” on page 527
● “COMMIT statement” on page 541
● “RELEASE SAVEPOINT statement” on page 931
● “ROLLBACK statement” on page 950
● “SAVEPOINT statement” on page 954
● “Savepoints within transactions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
● SQL/2008 ROLLBACK TO SAVEPOINT is part of optional SQL language feature T271 of the SQL/2008 standard.

ROLLBACK TRANSACTION statement [T-SQL]
Cancels any changes made since a SAVE TRANSACTION.

Syntax
ROLLBACK TRANSACTION [ savepoint-name ]
Remarks
The ROLLBACK TRANSACTION statement undoes any changes that have been made since a savepoint was established using SAVE TRANSACTION. Changes made before the SAVE TRANSACTION are not undone; they are still pending.

The savepoint-name is an identifier that was specified on a SAVE TRANSACTION statement within the current transaction. If savepoint-name is omitted, all outstanding changes are rolled back. Any savepoints since the named savepoint are automatically released.

There must be a corresponding SAVE TRANSACTION within the current transaction.

Privileges
None.

Side effects
None.

See also
- “ROLLBACK TO SAVEPOINT statement” on page 951
- “BEGIN TRANSACTION statement [T-SQL]” on page 527
- “COMMIT statement” on page 541
- “SAVE TRANSACTION statement [T-SQL]” on page 953

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example displays five rows with values 10, 20, and so on. The effect of the DELETE, but not the prior INSERTs or UPDATE, is undone by the ROLLBACK TRANSACTION statement.

```
BEGIN
    SELECT row_num INTO #tmp
    FROM sa_rowgenerator( 1, 5 )
    UPDATE #tmp SET row_num=row_num*10
    SAVE TRANSACTION before_delete
    DELETE FROM #tmp WHERE row_num >= 3
    ROLLBACK TRANSACTION before_delete
    SELECT * FROM #tmp
END
```

ROLLBACK TRIGGER statement
Undoes any changes made in a trigger.

Syntax
```
ROLLBACK TRIGGER [ WITH raiserror-statement ]
```
Remarks
The ROLLBACK TRIGGER statement rolls back the work done in a trigger, including the data manipulation that caused the trigger to fire.

Optionally, a RAISERROR statement can executed. If a RAISERROR statement is executed, an error is returned to the application. If no RAISERROR statement is executed, no error is returned.

If a ROLLBACK TRIGGER statement is used within a nested trigger and without a RAISERROR statement, only the innermost trigger and the statement which caused it to fire are undone.

Privileges
None.

Side effects
None

See also
- “CREATE TRIGGER statement” on page 713
- “ROLLBACK statement” on page 950
- “ROLLBACK TO SAVEPOINT statement” on page 951
- “RAISERROR statement” on page 920

Standards and compatibility
- SQL/2008 Vendor extension.
- Transact-SQL ROLLBACK TRIGGER is supported in both Watcom SQL and Transact-SQL stored procedures. ROLLBACK TRIGGER is supported by Adaptive Server Enterprise.

SAVE TRANSACTION statement [T-SQL]
Establishes a savepoint within the current transaction.

Syntax
SAVE TRANSACTION savepoint-name

Remarks
Establish a savepoint within the current transaction. The savepoint-name is an identifier that can be used in a ROLLBACK TRANSACTION statement. All savepoints are automatically released when a transaction ends.

Privileges
None.

Side effects
None.
Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following example displays five rows with values 10, 20, and so on. The effect of the DELETE, but not the prior INSERTs or UPDATE, is undone by the ROLLBACK TRANSACTION statement.

```
BEGIN
    SELECT row_num INTO #tmp
    FROM sa_rowgenerator( 1, 5 )
    UPDATE #tmp SET row_num=row_num*10
    SAVE TRANSACTION before_delete
    DELETE FROM #tmp WHERE row_num >= 3
    ROLLBACK TRANSACTION before_delete
    SELECT * FROM #tmp
END
```

**SAVEPOINT statement**

Establishes a savepoint within the current transaction.

**Syntax**

```
SAVEPOINT [ savepoint-name ]
```

**Remarks**

Establish a savepoint within the current transaction. The `savepoint-name` is an identifier that can be used in a RELEASE SAVEPOINT or ROLLBACK TO SAVEPOINT statement. All savepoints are automatically released when a transaction ends.

Savepoints that are established while a trigger or atomic compound statement is executing are automatically released when the atomic operation ends.

You cannot modify data in a proxy table from within a savepoint.

**Privileges**

None.

**Side effects**

None.
See also

- “RELEASE SAVEPOINT statement” on page 931
- “ROLLBACK TO SAVEPOINT statement” on page 951
- “SAVE TRANSACTION statement [T-SQL]” on page 953
- “Savepoints within transactions” [SQL Anywhere Server - SQL Usage]

Standards and compatibility

- SQL/2008  The SAVEPOINT statement is part of optional SQL/2008 language feature T271, "Savepoints".

- Transact-SQL  In Transact-SQL, creating a savepoint is accomplished with the SAVE TRANSACTION statement.

SELECT statement

Retrieves information from the database.

Syntax

```sql
[ WITH temporary-views ]
SELECT [ ALL | DISTINCT ] [ row-limitation-option-1 ] select-list
INTO { hostvar-list | variable-list | table-name }
INTO LOCAL TEMPORARY TABLE { table-name }
FROM from-expression
WHERE search-condition
GROUP BY group-by-expression
HAVING search-condition
WINDOW window-expression
ORDER BY { expression | integer } [ ASC | DESC ], ...
FOR READ ONLY | for-update-clause
FOR XML xml-mode
FOR JSON json-mode
row-limitation-option-2
OPTION( query-hint, ... ) ]
```

temporary-views :
regular-view, ...
| RECURSIVE { regular-view | recursive-view }, ...

regular-view :
view-name [ ( column-name, ... ) ]
AS ( subquery )

recursive-view :
view-name ( column-name, ... )
AS ( initial-subquery UNION ALL recursive-subquery )

row-limitation-option-1 :
FIRST
| TOP { ALL | limit-expression } [ START AT startat-expression ]

row-limitation-option-2 :
LIMIT { [ offset-expression, ] limit-expression | limit-expression OFFSET offset-expression }
limit-expression : simple-expression

startat-expression : simple-expression

offset-expression : simple-expression

simple-expression :
integer
| variable
| ( simple-expression )
| ( simple-expression { + | - | * } simple-expression )

select-list :
expression [ [ AS ] alias-name ], ...
| *
| window-function OVER { window-name | window-spec }
| [ [ AS ] alias-name ]
| sequence-expression

sequence-expression
sequence-name [ CURRVAL | NEXTVAL ]
FROM table-name

sequence-expression : See “Expressions” on page 21.

from-expression : See “FROM clause” on page 810.

group-by-expression : See “GROUP BY clause” on page 847.

search-condition : See “Search conditions” on page 40.

window-name : identifier

window-expression : See “WINDOW clause” on page 1051.

window-spec : See “WINDOW clause” on page 1051.

window-function :
RANK ( )
| DENSE_RANK ( )
| PERCENT_RANK ( )
| CUME_DIST ( )
| ROW_NUMBER ( )
| aggregate-function

for-update-clause
FOR UPDATE
| FOR UPDATE cursor-concurrency
| FOR UPDATE OF [ ( column-name, ... ) ]

cursor-concurrency :
BY { VALUES | TIMESTAMP | LOCK }

xml-mode :
RAW [ , ELEMENTS ]
AUTO [ , ELEMENTS ]
EXPLICIT

query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
FORCE OPTIMIZATION
FORCE NO OPTIMIZATION
option-name = option-value

option-name : identifier

option-value :
hostvar (indicator allowed)
| string
| identifier
| number

Parameters

WITH or WITH RECURSIVE clause  Define one or more common table expressions, also known as temporary views, to be used elsewhere in the remainder of the statement. These expressions may be non-recursive, or may be self-recursive. Recursive common table expressions may appear alone, or intermixed with non-recursive table expressions, only if the RECURSIVE keyword is specified. Mutually recursive common table expressions are not supported.

This clause is permitted only if the SELECT query block appears in one of the following locations:

○ Within a top-level SELECT query block including the top-level SELECT query block of a view definition

○ Within a top-level SELECT statement within an INSERT query block

○ Within a nested SELECT query block defining a derived table in any type of SQL statement

Recursive expressions consist of an initial subquery and a recursive subquery. The initial-query implicitly defines the schema of the view. The recursive subquery must contain a reference to the view within the FROM clause. During each iteration, this reference refers only to the rows added to the view in the previous iteration. The reference must not appear on the null-supplying side of an outer join. A recursive common table expression must not use aggregate functions and must not contain a GROUP BY, ORDER BY, or DISTINCT clause.

The WITH clause is not supported with remote tables. The WITH clause may also be used in INTERSECT, UNION, and EXCEPT query blocks.

ALL or DISTINCT clause  ALL (the default) returns all rows that satisfy the clauses of the SELECT statement. If DISTINCT is specified, duplicate output rows are eliminated. Many statements take significantly longer to execute when DISTINCT is specified, so you should reserve DISTINCT for cases where it is necessary.

row-limitation clauses  The row limitation clauses allow you to return only a subset of the rows that satisfy the WHERE clause. Only one row-limitation clause can be specified at a time. When specifying these clauses, an ORDER BY clause is required to order the rows in a meaningful manner.
○ **row-limitation-option-1**  The TOP and START AT arguments can be simple arithmetic expressions over host variables, integer constants, or integer variables. The TOP argument must evaluate to a value greater than or equal to 0. The START AT argument must evaluate to a value greater than 0.

If `startat-expression` is not specified, the default is 1. If the argument of TOP is ALL, all rows starting at `startat-expression` are returned. The TOP `limit-expression` START AT `startat-expression` clause is equivalent to LIMIT ( `startat-expression` -1 ), `limit-expression` or LIMIT `limit-expression` OFFSET ( `startat-expression` -1 ).

○ **row-limitation-option-2**  The LIMIT and OFFSET arguments can be simple arithmetic expressions over host variables, integer constants, or integer variables. The LIMIT argument must evaluate to a value greater than or equal to 0. The OFFSET argument must evaluate to a value greater than or equal to 0. If `offset-expression` is not specified, the default is 0.

The row limitation clause LIMIT `offset-expression`, `limit-expression` is equivalent to LIMIT `limit-expression` OFFSET `offset-expression`. Both of these constructs are equivalent to TOP `limit-expression` START AT ( `offset-expression` + 1 ).

The LIMIT keyword is disabled by default. Use the reserved_keywords option to enable the LIMIT keyword.

**select-list clause**  The **select-list** is a list of expressions, separated by commas, specifying what is retrieved from the database. An asterisk (*) means select all columns of all tables in the FROM clause.

Aggregate functions are allowed in the **select-list**. Subqueries are also allowed in the **select-list**. Each subquery must be within parentheses.

Alias names can be used throughout the query to represent the aliased expression.

Alias names are also displayed by Interactive SQL at the top of each column of output from the SELECT statement. If the optional alias name is not specified after an expression, Interactive SQL displays the expression.

**Note**
The following characters are not permitted in aliases:

○ Double quotes

○ Control characters (any character less than 0X20)

○ Backslashes

○ Square brackets

○ Back quotes

**INTO clause**  Following are the three uses of the INTO clause:
○ **INTO hostvar-list clause** This clause is used in embedded SQL only. It specifies where the results of the SELECT statement go. There must be one host variable item for each item in the *select-list*. *select-list* items are put into the host variables in order. An indicator host variable is also allowed with each host variable, so the program can tell if the *select-list* item was NULL.

If the query results in no rows being selected, the variables are not updated, and a row not found warning appears.

○ **INTO variable-list clause** This clause is used in procedures and triggers only. It specifies where the results of the SELECT statement go. There must be one variable for each item in the *select-list*. *select-list* items are put into the variables in order.

○ **INTO table-name clause** This clause is used to create a table and fill it with data.

For permanent tables to be created, the query must satisfy one of the following conditions:

- The *select-list* contains more than one item, and the INTO target is a single *table-name* identifier.
- The *select-list* contains a * and the INTO target is specified as *owner.table*.

To create a permanent table with one column, the table name must be specified as *owner.table*.

This statement causes a COMMIT before execution as a side effect of creating the table. No privileges are granted on the new table: the statement is a short form for CREATE TABLE followed by INSERT...SELECT.

Tables created using this clause do not have a primary key defined. You can add a primary key using ALTER TABLE. A primary key should be added before applying any updates or deletes to the table; otherwise, these operations result in all column values being logged in the transaction log for the affected rows.

**INTO LOCAL TEMPORARY TABLE** This clause is used to create a local, temporary table and populate it with the results of the query. When you use this clause, you do not need to start the temporary table name with #.

**FROM clause** Rows are retrieved from the tables and views specified in the *table-expression*. A SELECT statement with no FROM clause can be used to display the values of expressions not derived from tables. For example, these two statements are equivalent and display the value of the global variable @@version.

```sql
SELECT @@version;
SELECT @@version FROM DUMMY;
```

**WHERE clause** This clause specifies which rows are selected from the tables named in the FROM clause. It can be used to do joins between multiple tables, as an alternative to the ON phrase (which is part of the FROM clause).

**GROUP BY clause** You can group by columns, alias names, or functions. The result of the query contains one row for each distinct set of values in the named columns, aliases, or functions. As with DISTINCT and the set operations UNION, INTERSECT, and EXCEPT, the GROUP BY clause treats NULL values in the same manner as any other value in each domain. In other words, multiple NULL
values in a grouping attribute form a single group. Aggregate functions can then be applied to these
groups to get meaningful results.

When GROUP BY is used, the select-list, HAVING clause, and ORDER BY clause must not reference
any identifier that is not named in the GROUP BY clause. The exception is that the select-list and
HAVING clause can contain aggregate functions.

**HAVING clause**  This clause selects rows based on the group values and not on the individual row
values. The HAVING clause can only be used if either the statement has a GROUP BY clause or the
select-list consists solely of aggregate functions. Any column names referenced in the HAVING clause
must either be in the GROUP BY clause or be used as a parameter to an aggregate function in the
HAVING clause.

**WINDOW clause**  This clause defines all or part of a window for use with window functions such as
AVG and RANK.

**ORDER BY clause**  This clause sorts the results of a query. Each item in the ORDER BY list can be
labeled as ASC for ascending order (the default) or DESC for descending order. If the expression is an
integer \( n \), then the query results are sorted by the \( n \)th item in the select-list.

The only way to ensure that rows are returned in a particular order is to use ORDER BY. In the absence
of an ORDER BY clause, SQL Anywhere returns rows in whatever order is most efficient. The
appearance of result sets may vary depending on when you last accessed the row and other factors.

In embedded SQL, the SELECT statement is used for retrieving results from the database and placing the
values into host variables via the INTO clause. The SELECT statement must return only one row. For
multiple row queries, you must use cursors.

**FOR UPDATE or FOR READ ONLY clause**  These clauses specify whether updates are allowed
through a cursor opened on the query, and if so, what concurrency semantics can be used. This clause
cannot be used with the FOR XML clause.

If you do not use a FOR clause in the SELECT statement, the updatability of a cursor depends on the
cursor's declaration and how cursor concurrency is specified by the API. In ODBC, JDBC, OLE DB,
ADO.NET, and embedded SQL, statement updatability is explicit and a read-only cursor is used unless it
is overridden by the application. In Open Client and within stored procedures, cursor updatability does not
have to be specified, and the default is FOR UPDATE.

For Open Client and stored procedures, cursor updatability and statement updatability is dependent on the
setting of the ansi_update_constraints database option and the specific characteristics of the statement,
including whether the statement contains ORDER BY, DISTINCT, GROUP BY, HAVING, UNION,
aggregate functions, joins, or non-updatable views. For stored procedures, cursors default to FOR
UPDATE for single-table queries without an ORDER BY clause, or if the ansi_update_constraints option
is set to Off. When the ansi_update_constraints option is set to Cursors or Strict, then cursors over a query
containing an ORDER BY clause default to READ ONLY. However, you can explicitly mark cursors as
updatable using the FOR UPDATE clause. Because it is expensive to allow updates over cursors with an
ORDER BY clause or a join, cursors over a query containing a join of two or more tables are READ
ONLY and cannot be made updatable unless the ansi_update_constraints database option is Off.
A cursor declared FOR READ ONLY cannot be used in UPDATE (positioned), DELETE (positioned), or PUT statements. FOR READ ONLY is the default for embedded SQL.

The FOR UPDATE clause explicitly makes a cursor updatable. The use of FOR UPDATE alone does not, by itself, affect concurrency control on the rows in the result set of the statement. To do this, you must specify either FOR UPDATE BY LOCK or FOR UPDATE BY [ VALUES | TIMESTAMP ].

- **FOR UPDATE BY LOCK clause**  The database server acquires intent row locks on fetched rows of the result set. These are long-term locks that are held until the transaction is committed or rolled back. Intent row locks are not acquired if the SELECT statement uses an INTO clause since no positioned update can be performed.

- **FOR UPDATE BY { VALUES | TIMESTAMP }**  When you specify FOR UPDATE BY TIMESTAMP or FOR UPDATE BY VALUES, the database server uses optimistic concurrency by using a keyset-driven (value-sensitive) cursor. In this situation, lost updates can occur if the application modifies a row outside of the cursor (using a separate statement) or if the application does not heed the warnings and/or errors generated by the server indicating that the row has been modified by another connection.

To ensure that a statement acquires an intent lock, you must do one of the following:

- specify FOR UPDATE BY LOCK in the query
- specify HOLDLOCK, WITH ( HOLDLOCK ), WITH ( UPDLOCK ), or WITH ( XLOCK ) in the FROM clause of the query
- open the cursor with API calls that specify CONCUR_LOCK
- fetch the rows with attributes indicating fetch for update

The FOR UPDATE OF clause explicitly names the columns that can be modified with an UPDATE (positioned), DELETE (positioned), or PUT statement. You cannot use this clause in combination with any other FOR UPDATE, FOR READ ONLY, or FOR XML clause.

- **FOR UPDATE OF column-list clause**  When you specify the FOR UPDATE OF clause, the database server restricts the columns that can be modified with a positioned UPDATE or positioned DELETE statement to those columns that are explicitly named in that clause. An attempt to modify another column results in an error indicating that the column cannot be found. No check is made to determine if a column referenced within the list actually exists, or if that column's table is an updatable table.

**FOR XML clause**  This clause specifies that the result set is to be returned as an XML document. The format of the XML depends on the mode you specify. This clause cannot be used with the FOR UPDATE or FOR READ ONLY clause. Cursors declared with FOR XML are implicitly READ ONLY.

When you specify RAW mode, each row in the result set is represented as an XML <row> element, and each column is represented as an attribute of the <row> element.

AUTO mode returns the query results as nested XML elements. Each table referenced in the select-list is represented as an element in the XML. The order of nesting for the elements is based on the order that tables are referenced in the select-list.
EXPLICIT mode allows you to control the form of the generated XML document. Using EXPLICIT mode offers more flexibility in naming elements and specifying the nesting structure than either RAW or AUTO mode.

**FOR JSON clause**  This clause specifies that the result set is to be returned in JSON format. The JSON format depends on the mode you specify. This clause cannot be used with the FOR UPDATE or FOR READ ONLY clause. Cursors declared with FOR JSON are implicitly READ ONLY.

When you specify RAW mode, each row in the result set is returned as a flattened JSON representation.

AUTO mode returns the query results as nested JSON objects based on query joins.

EXPLICIT mode allows you to control the form of the generated JSON objects. Using EXPLICIT mode offers more flexibility in specifying columns and nested hierarchical objects to produce uniform or heterogeneous arrays.

**OPTION clause**  This clause provides hints about how to process the query. The following query hints are supported:

- **MATERIALIZED VIEW OPTIMIZATION clause**  Use the MATERIALIZED VIEW OPTIMIZATION clause to specify how the optimizer should make use of materialized views when processing the query. The specified option-value overrides the materialized_view_optimization database option for this query only. Possible values for option-value are the same values available for the materialized_view_optimization database option.

- **FORCE OPTIMIZATION clause**  When a query specification contains only simple queries (single-block, single-table queries that contain equality conditions in the WHERE clause that uniquely identify a specific row), it typically bypasses cost-based optimization during processing. Sometimes you may want cost-based optimization to occur. For example, if you want materialized views to be considered during query processing, view matching must occur. However, view matching only occurs during cost-based optimization. If you want cost-based optimization to occur for a query, but your query specification contains only simple queries, specify the FORCE OPTIMIZATION option to ensure that the optimizer performs cost-based optimization on the query.

  Similarly, specifying the FORCE OPTIMIZATION option in a SELECT statement inside of a procedure forces the use of the optimizer for any call to the procedure. In this case, plans for the statement are not cached.

- **FORCE NO OPTIMIZATION clause**  Specify the FORCE NO OPTIMIZATION clause if you want the statement to bypass the optimizer. If the statement is too complex to process in this way—possibly due to the setting of database options or characteristics of the schema or query—the statement fails and the database server returns an error.

- **option-name = option-value**  Specify an option setting. The setting you specify is only applicable to the current statement and takes precedence over any public or temporary option settings, including those set by ODBC-enabled applications.
The supported options are:

- “isolation_level option” [SQL Anywhere Server - Database Administration]
- “max_query_tasks option” [SQL Anywhere Server - Database Administration]
- “optimization_goal option” [SQL Anywhere Server - Database Administration]
- “optimization_level option” [SQL Anywhere Server - Database Administration]
- “optimization_workload option” [SQL Anywhere Server - Database Administration]
- “user_estimates option” [SQL Anywhere Server - Database Administration]

If you specify the isolation_level option in a query, the value specified in the query takes precedence over all other isolation level settings (such as setting the isolation_level option for the database or the setting for the cursor) for the current query.

sequence-expression You can select the current value (CURRVAL) or next value (NEXTVAL) from a sequence generator.

Remarks

The SELECT statement can be used:

- for retrieving results from the database.
- in Interactive SQL to browse data in the database, or to export data from the database to an external file.
- in procedures and triggers or in embedded SQL. A SELECT statement with an INTO clause is used for retrieving results from the database when the SELECT statement only returns one row. For multiple row queries, you must use cursors.
- to return a result set from a procedure.

Privileges

You must have the appropriate SELECT privileges on the objects referred to in the SELECT statement.

To select the CURRVAL or NEXTVAL values from a sequence generator, you must have USE ANY SEQUENCE system privilege, or be the owner of the sequence, or have been granted the required privileges to use the sequence generator.

Side effects

None.
Standards and compatibility

**SQL/2008**  Core feature. The complexity of the SELECT statement means that you should check individual clauses against the standard. For example, the ROLLUP keyword, which can be specified in a GROUP BY clause, is part of optional SQL/2008 language feature T431. Some of the SQL/2008 optional language features supported by SQL Anywhere include:

- The WINDOW clause and WINDOW aggregate functions comprise optional SQL/2008 language features T611 and T612.
- Sequence expressions are part of feature T176.
- Common table expressions are optional SQL/2008 language feature T121. A common table expression included in a nested query expression is feature T122. WITH RECURSIVE is optional SQL/2008 language feature T131; if included in a nested query it constitutes feature T132.
The ability to specify an ORDER BY clause with a query expression involving UNION, EXCEPT, or INTERSECT is optional feature F850. The ability to specify ORDER BY in a subquery is feature F851.

In the SQL standard, FOR UPDATE and FOR READ ONLY are part of a cursor declaration.

SQL Anywhere offers support for many extensions to the SQL/2008 definition of the SELECT statement. Some of these include:

- The optional cursor-concurrency clause (FOR UPDATE BY { LOCK | VALUES | TIMESTAMP}) is a vendor extension.
- The FOR XML, OPTION, and INTO clauses are vendor extensions.
- The row limitation clause is a vendor extension. In the SQL/2008 standard, row limitation is supported using FETCH FIRST syntax, which is optional language feature F856. The syntax for feature F856 is not supported by SQL Anywhere.
- The ability to specify ORDER BY $n$ is a vendor extension.
- In SQL/2008, all cursors except INSENSITIVE cursors are updatable by default. The read-only default with embedded SQL programs is a vendor extension.

**Transact-SQL**

There are substantial differences in SELECT statement support between SQL Anywhere and Adaptive Server Enterprise. Several features of the SELECT statement are not supported by Adaptive Server Enterprise.

These differences include:

- Sybase ASE does not support SQL Anywhere's cursor concurrency clause; to acquire a lock on a fetched row, you must use the HOLDLOCK table hint.
- Adaptive Server Enterprise does not support recursive queries or common table expressions.
- There are differences between Adaptive Server Enterprise and SQL Anywhere with respect to Transact-SQL outer joins.

In Transact-SQL you use the SELECT statement to assign a value to a variable, rather than with the Watcom SQL SET statement.

**Example**

This query returns the total number of employees in the Employees table.

```sql
SELECT COUNT(*)
FROM GROUPO.Employees;
```

This query lists all customers and the total value of their orders.

```sql
SELECT CompanyName,
       CAST( SUM( SalesOrderItems.Quantity * Products.UnitPrice ) AS INTEGER ) AS VALUE
FROM GROUPO.Customers
```
The following statement shows an embedded SQL SELECT statement where the number of employees in the Employees table is selected into the :size host variable:

```sql
SELECT COUNT(*) INTO :size
FROM GROUPO.Employees;
```

The following statement is optimized to return the first row in the result set quickly:

```sql
SELECT Name
FROM GROUPO.Products
GROUP BY Name
HAVING COUNT(*) > 1
AND MAX(UnitPrice) > 10
OPTION(optimization_goal='first-row');
```

### SET CONNECTION statement [Interactive SQL] [ESQL]

Changes the active database connection.

#### Syntax

```
SET CONNECTION [ connection-name ]
```

- `connection-name`: identifier | string | hostvar

#### Remarks

The SET CONNECTION statement changes the active database connection to connection-name. The current connection state is saved, and is resumed when it again becomes the active connection. If connection-name is omitted and there is a connection that was not named, that connection becomes the active connection.

When cursors are opened in embedded SQL, they are associated with the current connection. When the connection is changed, the cursor names of the previously active connection become inaccessible. These cursors remain active and in position, and become accessible when the associated connection becomes active again.

This SQL statement is not supported for SAP HANA databases.

#### Privileges

None.

#### Side effects

None.
See also
● “CONNECT statement [ESQL] [Interactive SQL]” on page 543
● “DISCONNECT statement [ESQL] [Interactive SQL]” on page 751
● “Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility
● SQL/2008  SET CONNECTION is part of optional SQL/2008 language feature F771, “Connection management”. Its usage within an Interactive SQL session is a vendor extension.

Example
The following example fragment is embedded SQL.

EXEC SQL SET CONNECTION :conn_name;

From Interactive SQL, set the current connection to the fictitious connection conn1.

SET CONNECTION conn1;

SET DESCRIPTOR statement [ESQL]

Describes the variables in a SQL descriptor area and to place data into the descriptor area.

Syntax

```
SET DESCRIPTOR descriptor-name
{ COUNT = { integer | hostvar } |
  VALUE { integer | hostvar } assignment, ... }

assignment :
{ TYPE | SCALE | PRECISION | LENGTH | INDICATOR }
  = { integer | hostvar }
| DATA = hostvar
```

descriptor-name : identifier

Remarks

The SET DESCRIPTOR statement is used to describe the variables in a descriptor area, and to place data into the descriptor area.

The SET...COUNT statement sets the number of described variables within the descriptor area. The value for count must not exceed the number of variables specified when the descriptor area was allocated.

The value { integer | hostvar } specifies the variable in the descriptor area upon which the assignment(s) is performed.

Type checking is performed when doing SET...DATA, to ensure that the variable in the descriptor area has the same type as the host variable. LONG VARCHAR and LONG BINARY are not supported by SET DESCRIPTOR...DATA.

If an error occurs, the code is returned in the SQLCA.
Privileges

None.

Side effects

None.

See also

- “ALLOCATE DESCRIPTOR statement [ESQL]” on page 424
- “DEALLOCATE DESCRIPTOR statement [ESQL]” on page 727
- “The SQL descriptor area (SQLDA)” [SQL Anywhere Server - Programming]

Standards and compatibility

- **SQL/2008** SET DESCRIPTOR is part of optional SQL/2008 language feature B031, "Basic dynamic SQL".

Example

The following example sets the type of the column with position col_num in sqlda.

```c
void set_type( SQLDA *sqlda, int col_num, int new_type )
{
    EXEC SQL BEGIN DECLARE SECTION;
    INT new_type1 = new_type;
    INT col = col_num;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SET DESCRIPTOR sqlda VALUE :col TYPE = :new_type1;
}
```

SET MIRROR OPTION statement

**Note**

Read-only scale-out and database mirroring each require a separate license. See “Separately licensed components” [*SQL Anywhere 16 - Introduction*].

Changes the values of options that control the settings for database mirroring and read-only scale-out.

Syntax

```
SET MIRROR OPTION option-name=\{ option-value \| NULL \}
```

```c
option-name :
    authentication_string
    auto_add_fan_out
    auto_add_server
    auto_failover
    child_creation
    page_timeout
    max_disconnected_time
    max_retry_connect_time
    synchronization_mode
```
Parameters

**NULL** Specifies the default value for the option. When the `option-name` is set to NULL, the option value is set to its default value.

<table>
<thead>
<tr>
<th><code>option-name</code></th>
<th>Applies to</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authentication_string</td>
<td>database mirroring</td>
<td>string</td>
<td>null</td>
<td>Specifies the authentication string used by all the servers in the database mirroring system. The authentication string is required for database mirroring.</td>
</tr>
<tr>
<td>auto_add_fan_out</td>
<td>read-only scale-out</td>
<td>integer</td>
<td>10</td>
<td>Specifies the maximum number of children for each branch. The minimum value that can be specified is 2.</td>
</tr>
<tr>
<td>auto_add_server</td>
<td>read-only scale-out</td>
<td>string</td>
<td>null</td>
<td>Specifies the name of the database server that acts as the parent of the automatic assignment tree.</td>
</tr>
<tr>
<td>auto_failover</td>
<td>database mirroring</td>
<td>on, off</td>
<td>null</td>
<td>Specifies whether the mirror server automatically takes over as the primary server when the current primary server goes down. This option does not apply to synchronous mode. This option accepts Boolean values (automatic failover is turned on with YES, ON, TRUE, or 1, and is turned off with any of NO, OFF, FALSE, and 0). The parameters are case insensitive. If you are using asynchronous or asyncfullpage mode, it is recommended that you set the auto_failover option to on. Then, if the primary server goes down, the mirror server automatically takes over as the primary server.</td>
</tr>
<tr>
<td><strong>option-name</strong></td>
<td>Applies to</td>
<td>Values</td>
<td>Default</td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
<td>---------------------------------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>child_creation</td>
<td>read-only</td>
<td>automatic, off, manual</td>
<td>automatic</td>
<td>Controls whether copy nodes are created automatically.</td>
</tr>
<tr>
<td>page_timeout</td>
<td>database</td>
<td>integer, in seconds</td>
<td>5</td>
<td>Specifies how often, in seconds, transaction log pages are sent to the mirror server, whether or not they are full. This option applies only when using asynchronous page mode.</td>
</tr>
<tr>
<td>max_disconnected_time</td>
<td>read-only</td>
<td>integer, in seconds, greater than or equal to max_retry_connect_time</td>
<td>no time limit</td>
<td>Specifies the amount of time since the last time the copy node was connected to the parent, alternate parent, or root database before the copy node shuts down.</td>
</tr>
<tr>
<td>max_retry_connect_time</td>
<td>read-only</td>
<td>integer, in seconds</td>
<td>120</td>
<td>Specifies the length of time that a copy node attempts to reconnect to its parent once the parent becomes unavailable.</td>
</tr>
<tr>
<td>promotion_time</td>
<td>read-only</td>
<td>integer, in seconds, greater than or equal to max_retry_connect_time</td>
<td>3600</td>
<td>Specifies the length of time that a copy node stays connected to the root database server after a parent connection is lost before promoting itself (adjusting the scale-out tree to not have a disconnected parent). This option helps to avoid adjusting the scale-out tree if a copy node is down for a short time (which can result in a shallow scale-out tree), while attempting to avoid increased load on the primary database for long periods of time. To never promote, set this option to 315,360,000 (10 years) or higher. This option is only supported in version 16 databases.</td>
</tr>
<tr>
<td><strong>option-name</strong></td>
<td>Applies to</td>
<td>Values</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>---------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>synchronization_mode</td>
<td>database mirroring</td>
<td>synchronous, asynchronous, asyncfull-page</td>
<td>synchronous</td>
<td>Specifies the synchronization mode used for database mirroring: synchronous (sync), asynchronous (async), or asyncfull-page (page). The synchronization mode controls when and how transactions are recorded on the mirror server.</td>
</tr>
</tbody>
</table>

**Remarks**

Once you create a database server for a database mirroring system or a read-only scale-out system using the CREATE MIRROR SERVER statement, you can use the SET MIRROR OPTION statement to configure the settings for the system.

**Privileges**

You must have started the server, or have the MANAGE ANY MIRROR SERVER system privilege.

**Side effects**

Automatic commit.

**See also**

- “Automatically assign the parent of a copy node” [SQL Anywhere Server - Database Administration]
- “How read-only scale-out systems handle the loss of a parent connection” [SQL Anywhere Server - Database Administration]
- “Database mirroring modes” [SQL Anywhere Server - Database Administration]
- “How child copy nodes are added” [SQL Anywhere Server - Database Administration]
- “-xa database server option” [SQL Anywhere Server - Database Administration]
- “Database mirroring” [SQL Anywhere Server - Database Administration]
- “SQL Anywhere read-only scale-out” [SQL Anywhere Server - Database Administration]
- “CREATE MIRROR SERVER statement” on page 615
- “ALTER MIRROR SERVER statement” on page 452
- “DROP MIRROR SERVER statement” on page 764
- “SYSMIRROROPTION system view” on page 1372

**Standards and compatibility**

- SQL/2008 Vendor extension.

**Example**

The following statement sets the authentication string for a database mirroring system to abc:

```
SET MIRROR OPTION authentication_string = 'abc';
```
SET OPTION statement

Changes the values of database and connection options.

Syntax 1

```
SET [ EXISTING ] [ TEMPORARY ] OPTION
[ userid | PUBLIC ] option-name = option-value
```

Syntax 2 (deprecated)

```
SET [ EXISTING ] [ TEMPORARY ] OPTION
[ userid | PUBLIC ] option-name = identifier
```

```
userid : identifier
option-name : identifier
option-value : ON, OFF, NULL, string literal, number, hostvar, or @variable-name
```

Parameters

- **option-value**  With Syntax 1, *option-value* can be one of:
  - the keywords ON, OFF, or NULL
  - a string literal value, within single quotation marks
  - a number of any valid format, including NUMERIC
  - within an embedded SQL program, the value of a host variable hostvar
  - the value of a SQL variable with a variable name that must begin with an @ sign

With Syntax 2, you can specify any valid identifier as an option value. Also, the database server treats the name of the identifier as if it were a string literal enclosed within single quotes. For example, the following two statements are equivalent:

```
SET TEMPORARY OPTION ansi_update_constraints = 'strict';
SET TEMPORARY OPTION ansi_update_constraints = strict;
```

Remarks

The SET OPTION statement is used to change options that affect the behavior of the database server. Setting the value of an option can change the behavior for all users (public), for an individual user, or for the current connection. The new setting can be made either temporary or permanent.

The classes of options that can be set with the SET OPTION statement are:

- Transact-SQL compatibility options.
- connection and database options.
- synchronization options.
- user-defined options

Option scope  With most options, you can set their value at three levels of scope: public, user, and connection. Some specific options, such as login_mode, are restricted to the public level only. A connection option takes precedence over the other two levels, and user options take precedence over
public options. You set a connection-level option by using the TEMPORARY keyword. If you set a user-
level option for the current user, the corresponding connection-level option is set at the same time.

By default, the option value applies to the currently logged on user ID that executed the SET OPTION
statement. If you specify a user ID, the option value applies to that user. If you specify PUBLIC, the
option value applies to all users who do not have an individual setting for the option.

**TEMPORARY options**  By default, a new option value is made permanent unless the TEMPORARY
keyword is specified. Adding the TEMPORARY keyword to the SET OPTION statement affects the
duration of the change.

When the SET TEMPORARY OPTION statement is not qualified with a user ID, the new option value is
in effect only for the current connection.

When SET TEMPORARY OPTION is used for the PUBLIC role, the change is in place for as long as the
database is running. When the database is shut down, TEMPORARY options for the PUBLIC role revert
back to their permanent value.

Setting temporary options for the PUBLIC role offers a security benefit. For example, when the
login_mode option is enabled, the database relies on the login security of the system on which it is
running. Enabling it temporarily means that a database relying on the security of a Windows domain is
not compromised if the database is shut down and copied to a local computer. In that case, the temporary
enabling of the login_mode option reverts to its permanent value, which could be Standard, a mode where
integrated logins are not permitted.

**Removing option settings**  If option-value is omitted, the specified option setting is deleted from the
database. If it was a user-level option setting, the value reverts back to the PUBLIC setting. If a
TEMPORARY option is deleted, the option setting reverts back to the permanent setting for that user.

**Option data types**  Options can have Boolean, numeric, or string values, but are always stored as
strings in the database. Option settings are always returned as strings as the result of a property function
or when returned as a result of a function or system stored procedure. Option values cannot be larger than
the database page size.

**User-defined options**  Any option, whether user-defined or not, must have a public setting before a
user-specific value can be assigned. The database server does not support setting TEMPORARY values
for user-defined options. For example, to create a user-defined option named ApplicationControl, you first
execute the statement:

```
SET OPTION PUBLIC.ApplicationControl = 'Default';
```

This statement sets the ApplicationControl option to Default for all users, and takes effect with each new
connection to the server. Subsequently, an individual user may establish their own setting for this option
by executing a separate SET OPTION statement.

**Restrictions**  If you use the EXISTING keyword, option values cannot be set for an individual user ID
unless there is already a PUBLIC setting for that option.
Caution
Do not change option values while a cursor is open. Changing the option values while a cursor is open can lead to inconsistent results within the cursor. For example, changing the date_format option while a cursor is open can result in some rows being returned in the old format and some rows returned in the new format. To ensure that the rows in the result set are computed consistently using the new option value, open the cursor after the option value is changed.

There are several ways you can query the value of specific options for a connection or user.

The SET OPTION statement is ignored by the SQL Flagger.

Privileges
Any user can set their own options.

To set database options for other users or roles, including the PUBLIC role, you must have one of the following system privileges, depending upon which privilege the option requires:

- SET ANY SYSTEM OPTION
- SET ANY PUBLIC OPTION
- SET ANY SECURITY OPTION
- SET ANY USER DEFINED OPTION

Side effects
Unless TEMPORARY is specified, an automatic commit is performed.

See also
- “Database options” [SQL Anywhere Server - Database Administration]
- “Alphabetical list of database options” [SQL Anywhere Server - Database Administration]
- “Remote ID settings” [MobiLink - Client Administration]
- “Compatibility options” [SQL Anywhere Server - Database Administration]
- “Synchronization options” [SQL Anywhere Server - Database Administration]
- “SQL Remote options” [SQL Anywhere Server - Database Administration]
- “How to view database options” [SQL Anywhere Server - Database Administration]
- “How to set database options using the SET OPTION statement” [SQL Anywhere Server - Database Administration]
- “SYSOPTION system view” on page 1375
- “sa_conn_options system procedure” on page 1110
- “sa_conn_options system procedure” on page 1110
- “CONNECTION_PROPERTY function [System]” on page 186
- “GET OPTION statement [ESQL]” on page 825
- “SET OPTION statement [Interactive SQL]” on page 975
- “SET statement [T-SQL]” on page 987
- “SET REMOTE OPTION statement [SQL Remote]” on page 978

Standards and compatibility
- SQL/2008 Vendor extension.
Example

Set the date format option for all users without an individual setting:

```
SET OPTION PUBLIC.date_format = 'Mmm dd yyyy';
```

Set the wait_for_commit option to On:

```
SET OPTION wait_for_commit = 'On';
```

The following fragment is an embedded SQL example:

```
EXEC SQL SET TEMPORARY OPTION date_format = :value;
```

Set the date_format option for the user that is currently connected. Future connections for the same user ID use this option value.

```
SET OPTION date_format = 'yyyy/mm/dd';
```

The following statement removes the setting of the date_format option for the current user ID. After executing this statement, the date_format setting for PUBLIC is used instead.

```
SET OPTION date_format=;
```

The following statement changes the login_mode option to Standard for grantees of the PUBLIC role:

```
SET OPTION PUBLIC.login_mode = 'Standard';
```

SET OPTION statement [Interactive SQL]

Changes the values of Interactive SQL options.

Syntax 1 - Set an Interactive SQL option

```
SET OPTION option-name = [ option-value ] | SET TEMPORARY OPTION option-name = [ option-value ]
```

```
option-name : identifier | string | hostvar

option-value : string | identifier | number
```

Syntax 2 - Save current Interactive SQL options permanently

```
SET PERMANENT
```

Syntax 3 - List current database option settings

```
SET
```

Remarks

When you set an option using the SET OPTION syntax, the option setting is stored permanently and does not change unless another SET OPTION statement changes it.

Using the SET TEMPORARY OPTION syntax allows you to temporarily change an option setting. The temporary setting remains in effect until you close Interactive SQL. The next time you start Interactive SQL, the option reverts to its permanent setting.
Use the SET PERMANENT syntax to permanently save all current Interactive SQL option settings (any temporary settings become permanent).

If option-value is omitted, the specified option is set to its default value.

Use Syntax 3 to display all the current database option settings in a window. If there are temporary options settings for the database server, they are displayed instead of the permanent settings.

Interactive SQL option settings are stored on the client computer, not in the database.

The following table lists the Interactive SQL options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>“auto_commit option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>On, Off</td>
<td>Off</td>
</tr>
<tr>
<td>“auto_refetch option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>On, Off</td>
<td>On</td>
</tr>
<tr>
<td>“bell option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>On, Off</td>
<td>On</td>
</tr>
<tr>
<td>“command_delimiter option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>String</td>
<td>','</td>
</tr>
<tr>
<td>“commit_on_exit option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>On, Off</td>
<td>On</td>
</tr>
<tr>
<td>“default_isql_encoding option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>String</td>
<td>Empty string</td>
</tr>
<tr>
<td>“echo option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>On, Off</td>
<td>On</td>
</tr>
<tr>
<td>“input_format option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>TEXT, FIXED</td>
<td>TEXT</td>
</tr>
<tr>
<td>“isql_allow_read_client_file option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>On, Off, Prompt</td>
<td>Prompt</td>
</tr>
<tr>
<td>Option</td>
<td>Values</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>“isql_allow_write_client_file option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>On, Off, Prompt</td>
<td>Prompt</td>
</tr>
<tr>
<td>“isql_command_timing option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>On, Off</td>
<td>On</td>
</tr>
<tr>
<td>“isql_escape_character option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>Character</td>
<td>‘\’</td>
</tr>
<tr>
<td>“isql_field_separator option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>String</td>
<td>‘,’</td>
</tr>
<tr>
<td>“isql_maximum_displayed_rows option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>All or a non-negative integer</td>
<td>500</td>
</tr>
<tr>
<td>“isql_print_result_set option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>Last, All, None</td>
<td>Last</td>
</tr>
<tr>
<td>“isql_quote option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>String</td>
<td>‘ ’</td>
</tr>
<tr>
<td>“isql_show_multiple_result_sets option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>On, Off</td>
<td>Off</td>
</tr>
<tr>
<td>“nulls option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>String</td>
<td>‘(NULL)’</td>
</tr>
<tr>
<td>“on_error option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>Stop, Continue, Prompt, Exit, Notify_Continue, Notify_Stop, Notify_Exit</td>
<td>Prompt</td>
</tr>
<tr>
<td>“output_format option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>TEXT, FIXED, HTML, SQL, XML</td>
<td>TEXT</td>
</tr>
<tr>
<td>“output_length option [Interactive SQL]” [SQL Anywhere Server - Database Administration]</td>
<td>Integer</td>
<td>0</td>
</tr>
<tr>
<td>Option</td>
<td>Values</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>“output_nulls option [Interactive SQL]”</td>
<td>String</td>
<td>Empty string</td>
</tr>
<tr>
<td>[SQL Anywhere Server - Database Administration]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>“truncation_length option [Interactive SQL]”</td>
<td>Integer</td>
<td>256</td>
</tr>
<tr>
<td>[SQL Anywhere Server - Database Administration]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Privileges

None.

Standards and compatibility

- SQL/2008  Vendor extension.

See also

- “Interactive SQL options” [SQL Anywhere Server - Database Administration]
- “Interactive SQL” [SQL Anywhere Server - Database Administration]

Example

The following example changes the value of the on_error option to continue:

```sql
SET OPTION on_error='continue';
```

SET REMOTE OPTION statement [SQL Remote]

Sets a message control parameter for a SQL Remote message link.

Syntax

```sql
SET REMOTE link-name OPTION
    [ userid.| PUBLIC.]link-option-name = link-option-value
```

*link-name* :
- file
- ftp
- http
- smtp

*link-option-name* :
- common-options
  - file-options
  - ftp-options
  - smtp-options

*common-options* :
- debug
- encode_dll
| max_retries  
| output_log_send_on_error  
| output_log_send_limit  
| output_log_send_now  
| pause_after_failure  

file-options :
  directory  
  | invalid_extensions  
  | unlink_delay  

ftp-options :
  active_mode  
  | host  
  | invalid_extensions  
  | password  
  | port  
  | root_directory  
  | reconnect_retries  
  | reconnect_pause  
  | suppress_dialogs  
  | user  

http-options :
  | certificate  
  | client_port  
  | https  
  | password  
  | proxy  
  | reconnect_retries  
  | reconnect_pause  
  | root_directory  
  | url  
  | user  

smtp-options :
  | local_host  
  | pop3_host  
  | pop3_password  
  | pop3_port  
  | pop3_userid  
  | smtp_authenticate  
  | smtp_option  
  | smtp_password  
  | smtp_port  
  | smtp_userid  
  | suppress_dialogs  
  | top_supported  

link-option-value : string

Parameters

userid  If you do not specify a userid, then the current publisher is assumed.

common-options  These options are common to the FILE, FTP, HTTP, and SMTP message systems:
debug  This parameter is set either to YES or NO. The default is NO. When set to YES, debug output specific to the message system is displayed. This information can be used for troubleshooting problems in the message system.

max_retries  By default, when SQL Remote is running in continuous mode and an error occurs when accessing the message system, it shuts down after the send and/or received phases. Use this parameter to specify the number of times you want SQL Remote to retry the send and/or receive phases before it shuts down.

output_log_send_on_error  Sends log information when an error occurs.

output_log_send_limit  Limits the amount of information that is sent to the consolidated database. The output_log_send_limit option specifies the number of bytes at the end of the output log (that is, the most recent entries) that are sent to the consolidated database. The default is 5K.

output_log_send_now  When set to YES, sends output log information to the consolidated database. On the next poll, the remote database sends the output log information and then resets the output_log_send_now option to NO.

pause_after_failure  This parameter applies when the max_retries parameter is specified to a value other than zero and SQL Remote is running in continuous mode. When an error occurs in the message system, this parameter defines the number of seconds SQL Remote waits between retrying the send and/or receive phases.

encode_dll  If you have implemented a custom encoding scheme, you must set this to the full path of the custom encoding DLL that you created.

file-options  These options apply to the FILE message system only:

directory  The directory under which the messages are stored. This parameter is an alternative to the SQLREMOTE environment variable.

invalid_extensions  A comma-separated list of file extensions that you do not want the SQL Remote Message Agent (dbremote) to use when generating files in the messaging system.

unlink_delay  The number of seconds to wait before attempting to delete a file if the previous attempt to delete the file failed. If no value is defined for unlink_delay, then the default behavior is set to pause for 1 second after the first failed attempt, 2 seconds after the second failed attempt, 3 seconds after the third failed attempt, and 4 seconds after the fourth failed attempt.

ftp-options  These options apply to the FTP message system only:

active_mode  This parameter controls how SQL Remote establishes the server/client connection. This parameter is set either to YES or NO. The default is NO (passive mode). Passive mode is the preferred transfer mode and the default for the FTP message link. In passive mode, all data transfer connections are initiated by the client, in this case, the message link. In active mode, the FTP server initiates all data connections.

host  The host name of the computer where the FTP server is running. This parameter can be a host name (such as FTP.ianywhere.com) or an IP address (such as 192.138.151.66).
○ **invalid_extensions**  A comma-separated list of file extensions that you do not want dbremote to use when generating files in the messaging system.

○ **password**  The password for accessing the FTP host.

○ **port**  The IP port number used for the FTP connection. This parameter is usually not required.

○ **reconnect_retries**  The number of times the link should try to open a socket with the server before failing. The default value is 4. When you set this parameter, only reconnections are affected. The initial connection made by the FTP link is not affected.

○ **reconnect_pause**  The time in seconds to pause between each connection attempt. The default setting is 30 seconds. When you set this parameter, only reconnections are affected. The initial connection made by the FTP link is not affected.

○ **root_directory**  The root directory within the FTP host site that the messages are stored under.

○ **suppress_dialogs**  This parameter is set to TRUE or FALSE. If it is set to TRUE, the Connect window does not appear after failed attempts to connect to the FTP server. Instead, an error is generated.

○ **user**  The user name for accessing the FTP host.

**http-options**  These options apply to the HTTP message system only:

○ **certificate**  To make a secure (HTTPS) request, a client must have access to the certificate used by the HTTPS server. The necessary information is specified in a string of semicolon-separated key/value pairs. You can use the file key to specify the file name of the certificate. You cannot specify a file and certificate key together. The following keys are available:

<table>
<thead>
<tr>
<th>Key</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td></td>
<td>The file name of the certificate</td>
</tr>
<tr>
<td>certificate</td>
<td>cert</td>
<td>The certificate itself</td>
</tr>
<tr>
<td>company</td>
<td>co</td>
<td>The company specified in the certificate</td>
</tr>
<tr>
<td>unit</td>
<td></td>
<td>The company unit specified in the certificate</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>The common name specified in the certificate</td>
</tr>
</tbody>
</table>

Certificates are required only for requests that are either directed to an HTTPS server or can be redirected from a non-secure to a secure server. Only PEM formatted certificates are supported.

```
certificate='file=filename'
```

To create a certificate name in a SQL Anywhere database:

```
CREATE OR REPLACE CERTIFICATE certificate_name FROM FILE 'certificate_file';
```
To use the certificate name for an HTTPS message type:

```
SET REMOTE HTTP OPTION user_name.certificate = 'cert_name=certificate_name';
```

- **client_port**  Identifies the port number on which SQL Remote communicates using HTTP. It is provided for, and recommended only for, connections through firewalls that filter "outgoing" TCP/IP connections. You can specify a single port number, ranges of port numbers, or a combination of the two. Specifying a low number of client ports could result in SQL Remote being unable to send and receive messages if the operating system has not released the ports in a timely manner after SQL Remote closes the port on a previous run.

- **debug**  When set to YES, all HTTP commands and responses are displayed in the output log. This information can be used for troubleshooting HTTP support problems. The default is NO.

- **https**  Specify whether to use HTTPS (https=yes) or HTTP (https=no).

- **password**  The message server database password. The password authenticates to third-party HTTP servers and gateways using RFC 2617 Basic authentication.

- **proxy_host**  Specifies the URI of a proxy server. For use when SQL Remote must access the network through a proxy server. Indicates that SQL Remote is to connect to the proxy server and send the request to the message server through it.

- **reconnect_retries**  The number of times the link should try to open a socket with the server before failing. The default value is 4. When you set this parameter, only reconnections are affected. The initial connection made by the FTP link is not affected.

- **reconnect_pause**  The time in seconds to pause between each connection attempt. The default setting is 30 seconds. When you set this parameter, only reconnections are affected. The initial connection made by the FTP link is not affected.

- **root_directory**  This HTTP control parameter is ignored when specified at the client side. You define this control parameter in the message server prior to calling the sr_add_message_server or sr_update_message_server stored procedure. When using the HTTP message system, the address specified for a remote user or publisher can only contain a single subdirectory, and not multiple subdirectories.

- **url**  Specify the server name or IP address and optionally the port number of the HTTP server being used, separated by a semicolon. If requests are being passed through the Relay Server, you can optionally add a URL extension to indicate which server farm the request should be passed to.

- **user**  The message server database user ID. Authenticates to third-party HTTP servers and gateways using RFC 2617 Basic authentication.

**smtp-options**  These options apply to the SMTP message system only:

- **local_host**  The name of the local computer. It is useful on computers where SQL Remote is unable to determine the local host name. The local host name is needed to initiate a session with any SMTP server. In most network environments, the local host name can be determined automatically and this entry is not needed.
- **pop3_host**  The name of the computer on which the POP host is running. Typically, it is the same name as the SMTP host. It corresponds to the POP3 host field in the SMTP/POP3 login window.

- **pop3_password**  The password used to retrieve mail. It corresponds to the password field in the SMTP/POP3 login window.

- **pop3_port**  The number of the port on which the POP server is listening. The default is 110. This corresponds to the port field in the SMTP/POP3 login window.

- **pop3_userid**  The user ID used to retrieve mail. The POP user ID corresponds to the user ID field in the SMTP/POP3 login window. You must obtain a user ID from your POP host administrator.

- **smtp_host**  The name of the computer on which the SMTP server is running. It corresponds to the SMTP host field in the SMTP/POP3 login window.

- **top_supported**  SQL Remote uses a POP3 command called TOP when enumerating incoming messages. The TOP command may not be supported by all POP servers. When you set the top_supported parameter to NO, SQL Remote uses the RETR command, which is less efficient but works with all POP servers. The default is YES.

- **smtp_authenticate**  Determines whether the SMTP link authenticates the user. The default value is YES. Set this parameter to NO to turn off SMTP authentication.

- **smtp_userid**  The user ID for SMTP authentication. By default, this parameter takes the same value as the pop3_userid parameter. The smtp_userid only needs to be set if the user ID is different from that of the POP server.

- **smtp_password**  The password for SMTP authentication. By default, this parameter takes the same value as the pop3_password parameter. The smtp_password only needs to be set if the user ID is different from that of the POP server.

- **smtp_port**  The number of the port on which the SMTP server is currently listening. The default is 25. This corresponds to the port field in the SMTP/POP3 login window.

- **suppress_dialogs**  When this parameter is set to true, the Connect window does not appear after failed attempts to connect to the mail server. Instead, an error is generated.

**Remarks**

The SQL Remote (dbremote) Message Agent saves message link parameters when the user enters them in the message link window when the message link is first used. In this case, it is not necessary to use this statement explicitly. This statement is most useful when preparing a consolidated database for extracting many databases.

The option names are case sensitive. The case sensitivity of option values depends on the option: Boolean values are case insensitive, while the case sensitivity of passwords, directory names, and other strings depend on the case sensitivity of the file system (for directory names), or the database (for user IDs and passwords).
Privileges

Publishers can set their own options. Otherwise, you must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

See also

- “Collecting errors from the remote database” [SQL Remote]
- “Setting remote message type control parameters” [SQL Remote]
- “Custom encoding schemes” [SQL Remote]
- “SET OPTION statement” on page 972
- “The FTP message system” [SQL Remote]
- “The FILE message system” [SQL Remote]
- “The HTTP message system” [SQL Remote]
- “Tutorial: Setting up a replication system using the HTTP message system” [SQL Remote]
- “The SMTP message system” [SQL Remote]
- “CREATE CERTIFICATE statement” on page 547
- “SET REMOTE OPTION statement [SQL Remote]” on page 978

Standards and compatibility

- SQL/2008 Vendor extension.

Examples

The following statement sets the FTP host to ftp.mycompany.com for the FTP link for user Sam_Singer:

```
SET REMOTE FTP OPTION Sam_Singer.host = 'ftp.mycompany.com';
```

The following statement stops SQL Remote from using the specified file extensions for messages that are generated:

```
SET REMOTE FTP OPTION "Public"."invalid_extensions"='exe,pif,dll,bat,cmd,vbs';
```

The following statement sets the URL to point to the localhost for the HTTP link for user Sam_Singer:

```
SET REMOTE HTTP OPTION Sam_Singer.url='localhost:8033';
```

The following statement sets the HTTP URL to point to a Relay Server that forwards the request to the srhttp farm:

```
SET REMOTE HTTP OPTION "public"."url"='iis7.company.com:80/rs/client/rs_client.dll/srhttp';
```

SET SQLCA statement [ESQL]

Instructs the SQL preprocessor to use a SQLCA other than the default, global sqlca.
Syntax

```
SET SQLCA sqlca
```

\[ sqlca \ : \ identifier \ | \ string \]

Remarks

The SET SQLCA statement tells the SQL preprocessor to use a SQLCA other than the default global sqlca. The sqlca must be an identifier or string that is a C language reference to a SQLCA pointer.

The current SQLCA pointer is implicitly passed to the database interface library on every embedded SQL statement. All embedded SQL statements that follow this statement in the C source file will use the new SQLCA.

This statement is necessary only when you are writing code that is reentrant.

The sqlca should reference a local variable. Any global or module static variable is subject to being modified by another thread.

Privileges

None.

Side effects

None.

See also

- “SQLCA management for multithreaded or reentrant code” [SQL Anywhere Server - Programming]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The owning function could be found in a Windows DLL. Each application that uses the DLL has its own SQLCA.

```
an_sql_code FAR PASCAL ExecuteSQL( an_application *app, char *com )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char *sqlcommand;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SET SQLCA "&app->.sqlca";
    sqlcommand = com;
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL EXECUTE IMMEDIATE :sqlcommand;
    return( SQLCODE );
}
```

**SET statement**

Assigns a value to a SQL variable.
Syntax

SET identifier = expression

Remarks

The SET statement assigns a new value to a variable. The variable must have been previously created using a CREATE VARIABLE statement or DECLARE statement, or it must be an OUTPUT parameter for a procedure. The variable name can optionally use the Transact-SQL convention of an @ sign preceding the name. For example: SET @localvar = 42

A variable can be used in a SQL statement anywhere a column name is allowed. If a column name exists with the same name as the variable, the variable value is used.

Variables are local to the current connection, and disappear when you disconnect from the database or use the DROP VARIABLE statement. They are not affected by COMMIT or ROLLBACK statements.

Variables are necessary for creating large text or binary objects for INSERT or UPDATE statements from embedded SQL programs because embedded SQL host variables are limited to 32767 bytes.

Privileges

None.

Side effects

None.

See also

- “CREATE VARIABLE statement” on page 722
- “DECLARE statement” on page 736
- “SET statement [T-SQL]” on page 987
- “DROP VARIABLE statement” on page 787
- “Expressions” on page 21
- “Host variable usage” [SQL Anywhere Server - Programming]

Standards and compatibility

- SQL/2008 The SET statement is part of optional SQL/2008 language feature P002, "Computational completeness".

- Transact-SQL The SET statement is supported by Adaptive Server Enterprise. In ASE, a single SET statement can be used to assign values to multiple variables, with individual assignment clauses separated by commas.

Example

This simple example shows the creation of a variable called birthday, and uses SET to set the date to CURRENT DATE.

```
CREATE VARIABLE @birthday DATE;
SET @birthday = CURRENT DATE;
```

The following code fragment inserts a large text value into the database.
EXEC SQL BEGIN DECLARE SECTION;
DECL_VARCHAR( 5000 ) buffer;
/* Note: maximum DECL_VARCHAR size is 32765 */
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE VARIABLE hold_blob LONG VARCHAR;
EXEC SQL SET hold_blob = '';
for(;;) {
    /* read some data into buffer ... */
    size = fread( buffer, 1, 5000, fp );
    if( size <= 0 ) break;
    /* Does not work if data contains null chars */
    EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES( 1, hold_blob );
EXEC SQL DROP VARIABLE hold_blob;

The following code fragment inserts a large binary value into the database.

EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY( 5000 ) buffer;
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE VARIABLE hold_blob LONG BINARY;
EXEC SQL SET hold_blob = '';
for(;;) {
    /* read some data into buffer ... */
    size = fread( &buffer.array, 1, 5000, fp );
    if( size <= 0 ) break;
    buffer.len = size;
    /* add data to blob using concatenation */
    EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES( 1, hold_blob );
EXEC SQL DROP VARIABLE hold_blob;

**SET statement [T-SQL]**

Sets database options for the current connection in an Adaptive Server Enterprise-compatible manner.

**Syntax**

```
SET option-name option-value
```

**Remarks**

The available options are as follows:

<table>
<thead>
<tr>
<th>Option name</th>
<th>Option value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ansi_null</td>
<td>On or Off</td>
</tr>
<tr>
<td>ansi_permissions</td>
<td>On or Off</td>
</tr>
<tr>
<td>close_on_endtrans</td>
<td>On or Off</td>
</tr>
</tbody>
</table>
The following options can be set using the Transact-SQL SET statement in SQL Anywhere and Adaptive Server Enterprise:

- **SET ansinull**  The default behavior for comparing values to NULL is different in SQL Anywhere and Adaptive Server Enterprise. Setting ansinull to Off provides Transact-SQL compatible comparisons with NULL.

  SQL Anywhere also supports the following syntax:

  ```sql
  SET ansi_nulls { On | Off }
  ```

- **SET ansi_permissions**  The default behavior is different in SQL Anywhere and Adaptive Server Enterprise regarding privileges required to perform an UPDATE or DELETE containing a column reference. Setting ansi_permissions to Off provides Transact-SQL-compatible privileges on UPDATE and DELETE.

- **SET close_on_endtrans**  The default behavior is different in SQL Anywhere and Adaptive Server Enterprise for closing cursors at the end of a transaction. Setting close_on_endtrans to Off provides Transact-SQL compatible behavior.

- **SET datefirst**  The default is 7, which means that the first day of the week is by default Sunday.

- **SET quoted_identifier**  Controls whether strings enclosed in double quotes are interpreted as identifiers (On) or as literal strings (Off).

- **SET rowcount**  `integer` The Transact-SQL ROWCOUNT option limits the number of rows fetched for any cursor to the specified integer. This includes rows fetched by re-positioning the cursor. Any
fetches beyond this maximum return a warning. The option setting is considered when returning the estimate of the number of rows for a cursor on an OPEN request.

SET ROWCOUNT also limits the number of rows affected by a searched UPDATE or DELETE statement to integer. This might be used, for example, to allow COMMIT statements to be performed at regular intervals to limit the size of the rollback log and lock table. The application (or procedure) would need to provide a loop to cause the update/delete to be re-issued for rows that are not affected by the first operation. A simple example is given below:

```
BEGIN
    DECLARE @count INTEGER
    SET rowcount 20
    WHILE(1=1) BEGIN
        UPDATE GROUP00.Employees SET Surname='new_name'
        WHERE Surname <> 'old_name'
        /* Stop when no rows changed */
        SELECT @count = @@rowcount
        IF @count = 0 BREAK
        PRINT string('Updated ',
                       @count,' rows; repeating...')
        COMMIT
    END
SET rowcount 0
END
```

If the ROWCOUNT setting is greater than the number of rows that Interactive SQL can display, Interactive SQL may do some extra fetches to reposition the cursor. So, the number of rows actually displayed may be less than the number requested. Also, if any rows are re-fetched due to truncation warnings, the count may be inaccurate.

A value of zero resets the option to get all rows.

- **SET self_recursion** The self_recursion option is used within triggers to enable (On) or prevent (Off) operations on the table associated with the trigger from firing other triggers.

- **SET string_rtruncation** The default behavior is different between SQL Anywhere and Adaptive Server Enterprise when non-space characters are truncated during assignment of SQL string data. Setting string_rtruncation to On provides Transact-SQL-compatible string comparisons.

- **SET textsize** Specifies the maximum size (in bytes) of TEXT or IMAGE type data to be returned with a SELECT statement. The @@textsize global variable stores the current setting. To reset to the default size (32 KB), use the command:

- **SET transaction isolation level** Sets the locking isolation level for the current connection.

  For Adaptive Server Enterprise, only 1 and 3 are valid options. For SQL Anywhere, any of 0, 1, 2, 3, snapshot, statement snapshot, and read only statement snapshot is a valid option.

  The SET statement is allowed by SQL Anywhere for the prefetch option, for compatibility, but has no effect.

**Privileges**

None.
Side effects

None.

See also

- “ansinull option” [SQL Anywhere Server - Database Administration]
- “Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]
- “Options for Transact-SQL compatibility” [SQL Anywhere Server - SQL Usage]
- “Compatibility options” [SQL Anywhere Server - Database Administration]
- “isolation_level option” [SQL Anywhere Server - Database Administration]
- “ansi_permissions option” [SQL Anywhere Server - Database Administration]
- “close_on_endtrans option” [SQL Anywhere Server - Database Administration]
- “quoted_identifier option” [SQL Anywhere Server - Database Administration]
- “string_rtruncation option” [SQL Anywhere Server - Database Administration]
- “first_day_of_week option” [SQL Anywhere Server - Database Administration]
- “DATEPART function [Date and time]” on page 208
- “SET OPTION statement” on page 972

Standards and compatibility

- SQL/2008  Transact-SQL extension.

SETUSER statement

Allows a user to assume the identity of (impersonate) another authorized user.

Syntax

```sql
{ SETUSER | SET SESSION AUTHORIZATION }
[ [ WITH OPTION ] userid ]
```

Parameters

**SETUSER or SET SESSION AUTHORIZATION**  SETUSER and SET SESSION AUTHORIZATION are semantically equivalent. However, the value you specify for SETUSER must be formatted as an identifier (for example, SETUSER JoeS or SETUSER "JoeS"), whereas the value you specify for SET SESSION AUTHORIZATION must be formatted as a string (for example, SETUSER 'JoeS').

**WITH OPTION clause**  During an impersonation session, database option settings in effect for the impersonator may be set differently than those of userid, which can impact results. Specify WITH OPTION to change the database options to reflect the options in effect for userid.

Remarks

The SETUSER statement is provided for administrative use and should not be used for connection pooling. After executing a SETUSER statement, you can execute one of the following commands to verify which user authorization you have assumed:

- SELECT USER
- SELECT CURRENT USER
SETUSER with no user ID undoes all earlier SETUSER statements.

A successful impersonation remains in effect until it is manually terminated (by executing a SETUSER statement with no ID) or the session is terminated.

The SETUSER statement cannot be used inside a procedure, trigger, event handler or batch.

There are several uses for the SETUSER statement, including the following:

- **Creating objects** You can use SETUSER to create a database object that is to be owned by another user.

- **Privilege checking** By acting as another user, with their privileges and inheritances, a user can test the privileges and name resolution of queries, procedures, views, and so on.

- **Providing a safer environment for administrators** The database administrator has permission to perform any action in the database. To ensure that you do not accidentally perform an unintended action, use SETUSER to switch to a different user ID with fewer privileges.

**Privileges**

You must have the SET USER system privilege. However, your ability to successfully execute a SETUSER statement (start an impersonation session) depends on whether you meet the at-least criteria for the person you are attempting to impersonate. The SETUSER statement fails if this condition is not met.

**See also**

- “Impersonation” [SQL Anywhere Server - Database Administration]
- “In-depth look at the impersonation at-least criteria” [SQL Anywhere Server - Database Administration]
- “EXECUTE IMMEDIATE statement [SP]” on page 791
- “GRANT statement” on page 827
- “REVOKE statement” on page 944
- “SET OPTION statement” on page 972
- “Identifiers” on page 4
- “Strings” on page 6

**Standards and compatibility**

- **SQL/2008** The SET SESSION AUTHORIZATION syntax is part of optional SQL/2008 language feature F321, "User authorization". The SETUSER syntax is a vendor extension. You can use the WITH OPTION syntax with both variants, but WITH OPTION is a vendor extension.

**Example**

In the first statement in this example (SETUSER "Joe"), a user who has the SET USER system privilege impersonates Joe to run some operations using Joe's privileges. In the second statement (SETUSER WITH OPTION "Jane"), the user impersonates Jane to perform some operations using Jane's privileges and the database options currently in effect for Jane. In the third statement (SETUSER), the user reverts back to their own user ID, privileges, and database options.
SETUSER "Joe"
// Some operations are run using Joes privileges ...
SETUSER WITH OPTION "Jane"
// Some operations are run using Jane's privileges, and the
// database options in effect are changed to the current
// database options for Jane
SETUSER

**SIGNAL statement [SP]**

Signals an exception condition.

**Syntax**

```
SIGNAL exception-name
```

**Remarks**

SIGNAL allows you to raise an exception.

Use `exception-name` to specify the name of an exception declared using a DECLARE statement at the beginning of the current compound statement. The exception must correspond to a system-defined SQLSTATE or a user-defined SQLSTATE. User-defined SQLSTATE values must be in the range 99000 to 99999.

**Privileges**

None.

**Side effects**

None.

**See also**

- “RESIGNAL statement [SP]” on page 936
- “BEGIN statement” on page 523
- “Exception handlers” [SQL Anywhere Server - SQL Usage]
- “RAISERROR statement” on page 920

**Standards and compatibility**

- **SQL/2008** The SIGNAL statement is part of optional SQL/2008 language feature P002, "Computational completeness".

- **Transact-SQL** The SIGNAL statement cannot be used in Transact-SQL compound statements and procedures.

**Example**

The following compound statement declares and signals a user-defined exception. If you execute this example from Interactive SQL, the message **My exception signaled** appears on the **Messages** tab in the **Results** area.
BEGIN
  DECLARE myexception EXCEPTION
  FOR SQLSTATE '99001';
  SIGNAL myexception;
  EXCEPTION
  WHEN myexception THEN
    MESSAGE 'My exception signaled'
    TO CLIENT;
END

START DATABASE statement

Starts a database on the current database server.

Syntax
START DATABASE database-file [ start-options ... ]

Parameters

database-file  The database-file parameter is a string. If a relative path is supplied in database-file, it is relative to the database server starting directory.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

start-options clause  The start-options clauses can be listed in any order:

  ○ AS clause  If database-name is not specified, a default name is assigned to the database. This default name is the root of the database file. For example, a database in file C:\Database Files \demo.db would be given the default name of demo. The database-name parameter is an identifier.

  ○ WITH TRUNCATE AT CHECKPOINT clause  Starts a database with log truncation on checkpoint enabled.

  ○ FOR READ ONLY clause  Starts a database in read-only mode. When used on a database requiring recovery, the statement fails and the error message is returned.

  ○ AUTOSTOP clause  The default setting for the AUTOSTOP clause is ON. With AUTOSTOP set to ON, the database is unloaded when the last connection to it is dropped. If AUTOSTOP is set to OFF, the database is not unloaded.

    In Interactive SQL, you can use YES or NO as alternatives to ON and OFF.
○ **KEY clause**   If the database is strongly encrypted, enter the KEY value (password) using this clause.

○ **WITH SERVER NAME clause**  Use this clause to specify an alternate name for the database server when connecting to this database.

**Note**
When using database mirroring, an alternate server name must be specified for client applications to be able to connect to the current primary server without knowing in advance which server is the primary server and which is the mirror server. Both partner servers must use the same name for the alternate server name. Use the CREATE MIRROR SERVER statement to create the alternate server names for the primary and mirror servers in a mirroring system.

○ **DIRECTORY clause**  Use this clause to specify the directory where the dbspace files are located for the database that is being started. For example, if the database server is started in the same directory as the dbspaces, and you include the DIRECTORY `'.'` clause, then this instructs the database server to find all dbspaces in the current directory.

**Note**
If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

○ **CHECKSUM clause**  Use this clause to enable write checksums for newly written pages for databases that were not created with global checksums enabled. This clause has the same behavior as the `-wc` database option.

The difference between the CHECKSUM clause and creating a database with global checksums enabled is that when you specify CHECKSUM ON, database pages are checksummed only when they are written out to disk. Pages that are read from disk are only verified if a checksum value was calculated before the pages were written. If a database has global checksums enabled, checksums are calculated for all pages when they are written and checksums are verified for all pages when they are read.

If the database server detects that the database is running on Windows Mobile or a removable storage device, such as a network share or USB device, then the database server automatically enables write checksums for all database pages.

By default, databases created with version 10 and 11 of SQL Anywhere do not have global checksums enabled. If you start a database created with SQL Anywhere 10 or 11 on a version 12 or later database server, then by default the database server creates write checksums for pages when they are written to disk (CHECKSUM ON). Version 12 and later databases have global checksums enabled by default, so the database server defaults to CHECKSUM OFF for these databases because by default all database pages have checksums. You can use either the `-wc` option or the START DATABASE statement to change the database server's checksum behavior if you do not want to use the default checksum settings.

You can check whether a database was created with global checksums enabled by executing the following statement:
You can check whether write checksums are enabled by executing the following statement:

```
SELECT DBPROPERTY('WriteChecksum');
```

- **DISKSANDBOX clause**  Set DISKSANDBOX to ON to restrict read-write file operations on the database to the directory where the main database file is located. Set DISKSANDBOX to OFF to allow access to all directories. If DISKSANDBOX is set to DEFAULT, the disk sandbox settings specified by the -sbx database server option are used. See “-sbx database server option” [SQL Anywhere Server - Database Administration].

**Note**

If you start a database server with the -sbx database server option, then you must provide the secure feature key for the manage_disk_sandbox secure feature to start a database with DISKSANDBOX OFF.

- **MIRROR ON clause**  Use the MIRROR ON clause to add an additional mirrored database to database servers that are already running and possibly hosting a mirrored database. You must specify the AUTOSTOP OFF clause when using this clause.

**Remarks**

Starts a specified database on the current database server.

The START DATABASE statement does not connect the current application to the specified database: an explicit connection is still needed.

If you are not connected to a database and you want to use the START DATABASE statement, you must first connect to a database, such as the utility database.

You can only use the database name utility_db to connect to the SQL Anywhere utility database.

**Privileges**

Anyone can start a database on the personal server (dbeng16).

The required privileges to start a database on a network server are specified by the database server -gd option. By default, the SERVER OPERATOR system privilege is required to start a database on the network server.

**Side effects**

None.

**Standards and compatibility**

- **SQL/2008**  Vendor extension.
See also

- “STOP DATABASE statement” on page 1004
- “CREATE MIRROR SERVER statement” on page 615
- “CONNECT statement [ESQL] [Interactive SQL]” on page 543
- “VALIDATE statement” on page 1045
- “-gd database server option” [SQL Anywhere Server - Database Administration]
- “-ds database option” [SQL Anywhere Server - Database Administration]
- “-ek database option” [SQL Anywhere Server - Database Administration]
- “-m database option” [SQL Anywhere Server - Database Administration]
- “-r database option” [SQL Anywhere Server - Database Administration]
- “-n database option” [SQL Anywhere Server - Database Administration]
- “-wc database option” [SQL Anywhere Server - Database Administration]
- “Setting up a database mirroring system” [SQL Anywhere Server - Database Administration]
- “The utility database” [SQL Anywhere Server - Database Administration]
- “Corruption detection using checksums” [SQL Anywhere Server - Database Administration]

Example

This example starts a fictitious database file `C:\temp\sample_2.db` on the current server:

```
START DATABASE 'c:\temp\sample_2.db';
```

This example starts same database but as sam2:

```
START DATABASE 'c:\temp\sample_2.db'
AS sam2;
```

### START SERVER statement [Interactive SQL]

Starts a database server.

**Syntax**

```
START SERVER AS database-server-name [ STARTLINE command-string ]
```

**Remarks**

The START SERVER statement starts a database server. To specify a set of options for the database server, use the STARTLINE keyword together with a command string.

START ENGINE is accepted for compatibility reasons, but is deprecated.

This SQL statement is not supported for SAP HANA databases.

**Privileges**

None

**Side effects**

None
See also

- “STOP SERVER statement” on page 1007
- “SQL Anywhere database server syntax” [SQL Anywhere Server - Database Administration]
- “Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

This example starts a database server named sample without starting any databases on it.

```
START SERVER AS sample;
```

This example shows the use of a STARTLINE clause.

```
START SERVER AS eng1 STARTLINE 'dbsrv16 -c 8M';
```

START EXTERNAL ENVIRONMENT statement

Starts an external environment.

Syntax

```
START EXTERNAL ENVIRONMENT environment-name
```

**environment-name**:  
JAVA  
PERL  
PHP  
CLR  
C_ESQL32  
C_ESQL64  
C_ODBC32  
C_ODBC64

Parameters

- **environment-name**  The name of the external environment to start.

Remarks

For more information about external environments, see “SQL Anywhere external environment support” [SQL Anywhere Server - Programming].

Privileges

None

Side effects

None
See also

- “SQL Anywhere external environment support” [SQL Anywhere Server - Programming]
- “ALTER EXTERNAL ENVIRONMENT statement” on page 438
- “STOP EXTERNAL ENVIRONMENT statement” on page 1005
- “INSTALL EXTERNAL OBJECT statement” on page 866
- “REMOVE EXTERNAL OBJECT statement” on page 933
- “SYSEXTERNENV system view” on page 1358
- “SYSEXTERNENVOBJECT system view” on page 1360

Standards and compatibility

- SQL/2008  Vendor extension.

Example

Start the Perl external environment.

```sql
START EXTERNAL ENVIRONMENT PERL;
```

**START JAVA statement**

Starts the Java VM.

**Syntax**

```sql
START JAVA
```

**Remarks**

The START JAVA statement starts the Java VM. The main use is to load the Java VM at a convenient time so that when the user starts to use Java functionality there is no initial pause while the Java VM is loaded.

The database server must be set up to locate a Java VM. Since you can specify different Java VMs for each database, the java_location option can be used to indicate the location (path) of the Java VM.

A Java VM must be installed.

This statement is not supported on Windows Mobile.

**Privileges**

None

**Side effects**

None

**See also**

- “STOP JAVA statement” on page 1006
- “How to start and stop the Java VM” [SQL Anywhere Server - Programming]
- “java_location option” [SQL Anywhere Server - Database Administration]
Standards and compatibility

- SQL/2008  Vendor extension.

Example

This example starts the Java VM.

```
START JAVA;
```

## START LOGGING statement [Interactive SQL]

Starts logging executed SQL statements to a log file.

### Syntax

```
START LOGGING filename
```

### Remarks

The START LOGGING statement starts copying all subsequent executed SQL statements to the log file that you specify. If the file does not exist, Interactive SQL creates it. Logging continues until you explicitly stop the logging process with the STOP LOGGING statement, or until you end the current Interactive SQL session.

You can also start and stop logging by clicking SQL » **Start Logging** and SQL » **Stop Logging**.

Execution times are included in the log file when logging and execution time reporting are both enabled.

### Privileges

None.

### Side effects

None.

### See also

- “STOP LOGGING statement [Interactive SQL]” on page 1007
- “`isql_command_timing` option [Interactive SQL]” [SQL Anywhere Server - Database Administration]
- “Logging statements in Interactive SQL” [SQL Anywhere Server - Database Administration]

### Standards and compatibility

- SQL/2008  Vendor extension.

### Example

This example starts logging to a file `c:\temp\filename.sql`:

```
START LOGGING 'c:\temp\filename.sql';
```
START SUBSCRIPTION statement [SQL Remote]

Starts a subscription for a user to a publication.

Syntax

```
START SUBSCRIPTION
TO publication-name [ ( subscription-value ) ]
FOR subscriber-id, ...
```

Parameters

- `publication-name` The name of the publication to which the user is being subscribed. This may include the owner of the publication.
- `subscription-value` A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.
- `subscriber-id` The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Remarks

A SQL Remote subscription starts when publication updates are being sent from the consolidated database to the remote database.

The START SUBSCRIPTION statement is one of a set of statements that manage subscriptions. The CREATE SUBSCRIPTION statement defines the data that the subscriber is to receive. The SYNCHRONIZE SUBSCRIPTION statement ensures that the consolidated and remote databases are consistent with each other. The START SUBSCRIPTION statement is required to start messages being sent to the subscriber.

Data at each end of the subscription must be consistent before a subscription is started. It is recommended that you use the database extraction utility to manage the creation, synchronization, and starting of subscriptions. If you use the database extraction utility, you do not need to execute an explicit START SUBSCRIPTION statement. Also, the Message Agent starts subscriptions once they are synchronized.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.
See also

- “CREATE SUBSCRIPTION statement [SQL Remote]” on page 683
- “REMOTE RESET statement [SQL Remote]” on page 932
- “SYNCHRONIZE SUBSCRIPTION statement [SQL Remote]” on page 1016
- “STOP SUBSCRIPTION statement [SQL Remote]” on page 1008
- “Extraction utility (dbxtract)” [SQL Remote]
- “Starting subscriptions” [SQL Remote]
- “Creating multiple remote databases” [SQL Remote]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement starts the subscription of user Sam_Singer to the pub_contact publication.

```
START SUBSCRIPTION TO pub_contact
FOR Sam_Singer;
```

START SYNCHRONIZATION DELETE statement [MobiLink]

Restarts logging of deletes for MobiLink synchronization.

Syntax

```
START SYNCHRONIZATION DELETE
```

Remarks

Ordinarily, SQL Anywhere and UltraLite automatically log any changes made to tables or columns that are part of a synchronization, and upload these changes to the consolidated database during the next synchronization. You can temporarily suspend automatic logging of delete operations using the STOP SYNCHRONIZATION DELETE statement. The START SYNCHRONIZATION DELETE statement allows you to restart the automatic logging.

When a STOP SYNCHRONIZATION DELETE statement is executed, none of the delete operations executed on that connection are synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed. Repeating STOP SYNCHRONIZATION DELETE has no additional effect.

A single START SYNCHRONIZATION DELETE statement restarts the logging, regardless of the number of STOP SYNCHRONIZATION DELETE statements preceding it.

Do not use START SYNCHRONIZATION DELETE if your application does not synchronize data.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

None.
See also

- “STOP SYNCHRONIZATION DELETE statement [MobiLink]” on page 1010
- “ULConnection.StartSynchronizationDelete method [UltraLite.NET]” [UltraLite - .NET Programming]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION DELETE and STOP SYNCHRONIZATION DELETE:

```
-- Prevent deletes from being sent
-- to the consolidated database
STOP SYNCHRONIZATION DELETE;

-- Remove all records older than 1 month
-- from the remote database,
-- NOT the consolidated database
DELETE FROM PROPOSAL
WHERE last_modified < months( CURRENT TIMESTAMP, -1 )

-- Re-enable all deletes to be sent
-- to the consolidated database
-- DO NOT FORGET to start this
START SYNCHRONIZATION DELETE;

-- Commit the entire operation,
-- otherwise rollback everything
-- including the stopping of the deletes
commit;
```

START SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]

Starts a MobiLink synchronization schema change.

Syntax

```
START SYNCHRONIZATION SCHEMA CHANGE
FOR TABLES table-list
set-script-version
  | set-script-version-on-subscription, ...

  set-script-version : 
  SET SCRIPT VERSION = script-version

  set-script-version-on-subscription : 
  SET SCRIPT VERSION = script-version ON SUBSCRIPTION subscription_name

  script-version : string | NULL

  subscription-name : identifier
```
Parameters

**FOR TABLES clause**  
This clause specifies the tables that are affected by the schema change.

**SET SCRIPT VERSION clause**  
Specifies the new script version for all subscriptions that contain any table specified in the FOR TABLES clause. The new script version may be the same as the existing script version.

**SET SCRIPT VERSION ... ON SUBSCRIPTION clause**  
Specifies the new script version for the specified subscription. When used, this clause must be repeated, separated by commas, for each subscription that contains any table specified in the FOR TABLES clause. The new script version may be the same as the existing script version.

Remarks

All tables to which you want to apply a schema change must be listed in `table-list`. A table cannot be listed more than once. An error message is reported if there is an existing lock on any of the tables in `table-list`.

Only one synchronization schema change can be executed on a database at a time. The START SYNCHRONIZATION SCHEMA CHANGE statement fails when another schema change is in progress.

The database server obtains locks on all tables specified in `table-list`. The database server ignores the setting of the blocking option when attempting to obtain locks. If a lock cannot be obtained, all previously acquired locks are released and an error message is reported.

During a synchronization schema change:

- You cannot execute a data manipulation statement.
- You cannot execute additional START SYNCHRONIZATION SCHEMA CHANGE statements.
- You can alter a publication to change the column subsetting of any table in `table-list`.
- You can alter a publication to drop any table in `table-list`.
- You can alter any of the tables listed in `table-list`.

An implicit commit is performed both before and after the START SYNCHRONIZATION SCHEMA CHANGE statement is executed. A synchronization schema change ends with the execution of a STOP SYNCHRONIZATION SCHEMA CHANGE statement. When the STOP SYNCHRONIZATION SCHEMA CHANGE statement is executed, all table locks are released.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

None.
See also

- “STOP SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]” on page 1015
- “SynchronizationSchemaChangeActive database property” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION SCHEMA CHANGE and STOP SYNCHRONIZATION SCHEMA CHANGE:

```
START SYNCHRONIZATION SCHEMA CHANGE
  FOR TABLES GROUPO.SalesOrders, GROUPO.Products
  SET SCRIPT VERSION = 'version_2' ON SUBSCRIPTION sub1,
  SET SCRIPT VERSION = 'version_2' ON SUBSCRIPTION sub2;
ALTER TABLE GROUPO.SalesOrders ADD SUBTOTAL NUMERIC (10,2);
ALTER TABLE GROUPO.Products ALTER QUANTITY BIGINT;
STOP SYNCHRONIZATION SCHEMA CHANGE;
```

STOP DATABASE statement

Stops a database on the current database server.

Syntax

```
STOP DATABASE database-name
  [ ON database-server-name ]
  [ UNCONDITIONALLY ]
```

Parameters

- **STOP DATABASE clause**  The `database-name` is the name of a database (other than the current database) running on the current server.

- **ON clause**  This clause is supported in Interactive SQL only. If `database-server-name` is not specified in Interactive SQL, all running servers are searched for a database of the specified name.

When not using this statement in Interactive SQL, the database is stopped only if it is started on the current database server.

- **UNCONDITIONALLY clause**  Stop the database even if there are connections to the database. By default, the database is not stopped if there are connections to it.

Remarks

The STOP DATABASE statement stops a specified database on the current database server.

You cannot use STOP DATABASE on the database to which you are currently connected.

Privileges

Anyone can stop a database on the personal server.
The required privileges to stop a database on the network server are determined by the database server -gd option. The default system privilege for stopping a database on the network server is SERVER OPERATOR.

Side effects
None

See also

- “START DATABASE statement” on page 993
- “DISCONNECT statement [ESQL] [Interactive SQL]” on page 751
- “-gd database server option” [SQL Anywhere Server - Database Administration]
- “Stop Server utility (dbstop)” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

Stop the database named sample on the current server.

STOP DATABASE sample;

STOP EXTERNAL ENVIRONMENT statement

Stops an external environment.

Syntax

STOP EXTERNAL ENVIRONMENT environment-name

environment-name:
  JAVA
  | PERL
  | PHP
  | CLR
  | C_ESQL32
  | C_ESQL64
  | C_ODBC32
  | C_ODBC64

Parameters

environment-name The name of the external environment to stop.

Remarks

For more information about external environments, see “SQL Anywhere external environment support” [SQL Anywhere Server - Programming].

Privileges

None
Side effects
None

See also
- “SQL Anywhere external environment support” [SQL Anywhere Server - Programming]
- “ALTER EXTERNAL ENVIRONMENT statement” on page 438
- “START EXTERNAL ENVIRONMENT statement” on page 997
- “INSTALL EXTERNAL OBJECT statement” on page 866
- “REMOVE EXTERNAL OBJECT statement” on page 933
- “SYSEXTERNENV system view” on page 1358
- “SYSEXTERNENVOBJECT system view” on page 1360

Standards and compatibility
- SQL/2008 Vendor extension.

Example
This example stops the Perl external environment.

STOP EXTERNAL ENVIRONMENT PERL;

STOP JAVA statement
Stops the Java VM.

Syntax
STOP JAVA

Remarks
The STOP JAVA statement unloads the Java VM when it is not in use. The main use is to economize on the use of system resources.

This statement is not supported on Windows Mobile.

Privileges
None.

Side effects
None

See also
- “START JAVA statement” on page 998

Standards and compatibility
- SQL/2008 Vendor extension.
Example

This example stops the Java VM.

```sql
STOP JAVA;
```

STOP LOGGING statement [Interactive SQL]

Stops logging of SQL statements in the current session.

Syntax

```sql
STOP LOGGING
```

Remarks

The STOP LOGGING statement stops Interactive SQL from writing each SQL statement you execute to a log file. You can start logging with the START LOGGING statement.

You can also stop logging by clicking SQL » Stop Logging.

Privileges

None.

Side effects

None.

See also

- “START LOGGING statement [Interactive SQL]” on page 999
- “Logging statements in Interactive SQL” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example stops the current logging session.

```sql
STOP LOGGING;
```

STOP SERVER statement

Stops a database server.

Syntax

```sql
STOP SERVER [ database-server-name ] [ UNCONDITIONALLY ]
```
 Parameters

**UNCONDITIONALLY clause** If you are the only connection to the database server, you do not need to use UNCONDITIONALLY. If there are other connections, the database server stops only if you use the UNCONDITIONALLY keyword.

 Remarks

You can only use *database-server-name* in Interactive SQL only. If you do not execute this statement from Interactive SQL, the current database server is stopped.

By default, the database server is not stopped if there are other connections to it. If the UNCONDITIONALLY clause is used, the database server is stopped even if there are other connections to the database server. If the STOP SERVER statement is executed on a client connection and the server successfully shuts down, a communication error occurs.

The STOP SERVER statement cannot be used in stored procedures, triggers, events, or batches.

STOP ENGINE is accepted for compatibility reasons, but is deprecated.

 Privileges

Anyone can shut down the personal server (dbeng16).

The privileges to shut down a network server (dbsrv16) depend on the -gk setting on the database server command line. By default, the SERVER OPERATOR privilege is required to shut down a network server.

 Side effects

None

 See also

- “START SERVER statement [Interactive SQL]” on page 996
- “-gk database server option” [SQL Anywhere Server - Database Administration]

 Standards and compatibility

- SQL/2008 Vendor extension.

 Example

Stop the current database server, as long as there are no other connections.

```
STOP SERVER;
```

STOP SUBSCRIPTION statement [SQL Remote]

Stops a subscription for a user to a publication.
Syntax

**STOP SUBSCRIPTION**
**TO publication-name** [ ( subscription-value ) ]
**FOR subscriber-id, ...**

Parameters

- **publication-name**  The name of the publication to which the user is being subscribed. This may include the owner of the publication.

- **subscription-value**  A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.

- **subscriber-id**  The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Remarks

A SQL Remote subscription starts when publication updates are being sent from the consolidated database to the remote database.

The STOP SUBSCRIPTION statement prevents any further messages being sent to the subscriber. The START SUBSCRIPTION statement is required to restart messages being sent to the subscriber. However, you should ensure that the subscription is properly synchronized before restarting: that no messages have been missed.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.

See also

- “DROP SUBSCRIPTION statement [SQL Remote]” on page 776
- “START SUBSCRIPTION statement [SQL Remote]” on page 1000
- “SYNCHRONIZE SUBSCRIPTION statement [SQL Remote]” on page 1016
- “Extraction utility (dbxtract)” [SQL Remote]
- “Stopping subscriptions” [SQL Remote]

Standards and compatibility

- **SQL/2008**  Vendor extension.

Example

The following statement stops the subscription of user SamS to the pub_contact publication.

```sql
STOP SUBSCRIPTION TO pub_contact
FOR Sam_Singer;
```
STOP SYNCHRONIZATION DELETE statement [MobiLink]

Temporarily stops logging of deletes for MobiLink synchronization.

Syntax

```
STOP SYNCHRONIZATION DELETE
```

Remarks

Ordinarily, SQL Anywhere and UltraLite remote databases automatically log any changes made to tables or columns that are being synchronized, and then upload these changes to the consolidated database during the next synchronization. This statement allows you to temporarily suspend logging of delete operations to a SQL Anywhere or UltraLite remote database.

None of the delete operations executed on a connection between the time the connection executes STOP SYNCHRONIZATION DELETE and the time the connection executes START SYNCHRONIZATION DELETE are synchronized.

A single START SYNCHRONIZATION DELETE statement restarts the logging, regardless of the number of STOP SYNCHRONIZATION DELETE statements preceding it.

This statement can be useful to make corrections to a remote database, but should be used with caution as it effectively disables MobiLink synchronization.

Do not use STOP SYNCHRONIZATION DELETE if your application does not synchronize data.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

None.

See also

- “ULConnection.StartSynchronizationDelete method [UltraLite.NET]” [UltraLite -.NET Programming]
- “ULConnection.StopSynchronizationDelete method [UltraLite.NET]” [UltraLite -.NET Programming]
- “START SYNCHRONIZATION DELETE statement [MobiLink]” on page 1001

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION DELETE and STOP SYNCHRONIZATION DELETE:

```
-- Prevent deletes from being sent
-- to the consolidated database
STOP SYNCHRONIZATION DELETE;
```
-- Remove all records older than 1 month
-- from the remote database,
-- NOT the consolidated database
DELETE FROM PROPOSAL
WHERE last_modified < months( CURRENT_TIMESTAMP, -1 )

-- Re-enable all deletes to be sent
-- to the consolidated database
-- DO NOT FORGET to start this
START SYNCHRONIZATION DELETE;

-- Commit the entire operation,
-- otherwise rollback everything
-- including the stopping of the deletes
commit;

SYNCHRONIZE statement [MobiLink]
Synchronizes a SQL Anywhere database with a MobiLink server. The synchronization options can be specified in the statement itself.

Syntax
SYNCHRONIZE {
PROFILE sync-profile-name [ MERGE sync-option [ ;... ] ]
| USING sync-option [ ;... ]
| START
| STOP
}

[ PORT port-number ]
[ VERBOSITY { LOW | NORMAL | HIGH } ]
[ TIMEOUT timeout ]
[ USER user-name IDENTIFIED BY password ]

sync-option : string

Parameters
sync-profile-name The name of the synchronization profile to use for this synchronization.

MERGE clause Use this clause to add or override synchronization profile options.

USING clause Use this clause to specify synchronization profile options when you are not using a synchronization profile.

sync-option A string of one or more synchronization profile option value pairs, separated by semicolons. For example, 'option1=value1;option2=value2'.

PORT clause Use this clause to specify the port number that the database server uses to communicate with the dbmlsync utility. The default is 4433.

VERBOSITY clause This clause controls the amount of information that is added to the synchronize_results and synchronize_parameters shared global temporary tables during synchronization.
The following is a list of client API events that are returned by each VERBOSITY option.

<table>
<thead>
<tr>
<th>Option</th>
<th>Returns</th>
</tr>
</thead>
</table>
| LOW              | ○ DBSC_EVENTTYPE_SYNC_START  
|                  | ○ DBSC_EVENTTYPE_SYNC_DONE  
|                  | ○ DBSC_EVENTTYPE_ERROR_MSG  
|                  | ○ DBSC_EVENTTYPE_WARNING_MSG  |
| NORMAL (default) | ○ DBSC_EVENTTYPE_SYNC_START  
|                  | ○ DBSC_EVENTTYPE_SYNC_DONE  
|                  | ○ DBSC_EVENTTYPE_ERROR_MSG  
|                  | ○ DBSC_EVENTTYPE_WARNING_MSG  
|                  | ○ DBSC_EVENTTYPE_INFO_MSG   |
| HIGH             | ○ DBSC_EVENTTYPE_SYNC_START  
|                  | ○ DBSC_EVENTTYPE_SYNC_DONE  
|                  | ○ DBSC_EVENTTYPE_ERROR_MSG  
|                  | ○ DBSC_EVENTTYPE_WARNING_MSG  
|                  | ○ DBSC_EVENTTYPE_INFO_MSG   
|                  | ○ DBSC_EVENTTYPE_PROGRESS_INDEX  
|                  | ○ DBSC_EVENTTYPE_PROGRESS_TEXT  
|                  | ○ DBSC_EVENTTYPE_TITLE       |

Be careful not to confuse the VERBOSITY clause of the SYNCHRONIZE statement with the VERBOSITY option that you can specify in a synchronization profile. The VERBOSITY clause of the SYNCHRONIZE statement controls the type of events that are recorded in the synchronize_results and synchronize_parameters tables. The VERBOSITY option in a synchronization profile controls the number of DBSC_EVENTTYPE_INFO_MSG events that are generated during synchronization.

```
SYNCHRONIZE PROFILE SalesData VERBOSITY NORMAL;
SYNCHRONIZE PROFILE SalesData MERGE 'Verbosity=BASIC,ROW_DATA' VERBOSITY NORMAL;
```

**TIMEOUT clause**  This clause specifies how long the database server waits, in seconds, for the synchronization to complete before attempting to cancel the synchronization. The default is 240 seconds.

**USER/IDENTIFIED BY clause**  Use this clause to specify the database user ID and password that the dbmlsync utility uses to synchronize the database. The user ID specified must have the SYS_RUN_REPLICATION_ROLE system role. By default, synchronization uses the user ID for the database connection that executed the SYNCHRONIZE statement.

**START clause**  Starts the dbmlsync utility running in server mode and leaves it running. No synchronization is performed. When you are performing more than one synchronization in a short period, you can improve performance by explicitly starting the dbmlsync server using this clause, performing your synchronizations, then explicitly stopping the dbmlsync server using the STOP clause.

**STOP clause**  Stops a dbmlsync server that was previously started using the START clause. No synchronization is performed.
Remarks
This statement can only be used if the MobiLink client for SQL Anywhere including the Dbmlsync C++ API is installed.

The MobiLink client for SQL Anywhere is not available on all platforms where the database server may run. For a list of supported platforms, see http://www.sybase.com/detail?id=1061806.

When synchronization is complete, you can view the results of the synchronization in the synchronize_results and synchronize_parameters shared global temporary tables. The synchronize_results and synchronize_parameters tables store the results of all synchronizations that have been executed with the SYNCHRONIZE statement since the database server was started. The synchronize_results and synchronize_parameters tables are truncated each time the database server is shut down.

The synchronize_results table contains the following columns:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>UNSIGNED BIGINT</td>
<td>The primary key of the table used to determine the order in which rows were inserted into the table.</td>
</tr>
<tr>
<td>conn_id</td>
<td>UNSIGNED INT</td>
<td>The connection id number of the connection that executed the SYNCHRONIZE statement that generated this event.</td>
</tr>
<tr>
<td>result_time</td>
<td>TIMESTAMP</td>
<td>The time the event was added to the synchronize_results table.</td>
</tr>
<tr>
<td>result_type</td>
<td>CHAR(128)</td>
<td>The type of event.</td>
</tr>
</tbody>
</table>

Each event shown in the synchronize_results table has 0 or more parameters associated with it that contain additional information about the event. The parameters are stored in the synchronize_parameters table, which contains the following columns:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>UNSIGNED BIGINT</td>
<td>A foreign key to the row_id column in the synchronize_results table. Use this value to match each parameter back to the event to which it belongs.</td>
</tr>
<tr>
<td>parm_id</td>
<td>UNSIGNED INT</td>
<td>Contains the numeric ID of the parameter. For events with more than 1 parameter, use this value to locate the specific parameter you need.</td>
</tr>
<tr>
<td>parm_message</td>
<td>LONG VARCHAR</td>
<td>The value associated with the parameter.</td>
</tr>
</tbody>
</table>
To view information about past or current synchronizations, you can use the sp_get_last_synchronize_result system procedure as an alternative to directly querying the synchronize_results and synchronize_parameters tables.

Alternately, you can use the following statement to view the results of all the synchronizations that have taken place since the database server was started.

```sql
SELECT * FROM synchronize_results sr
KEY JOIN synchronize_parameters sp
ORDER BY sr.row_id, sp.parm_id
```

You can use the synchronize_results and synchronize_parameters tables to monitor the progress of a synchronization on a connection that is different from your current connection. To monitor the progress of a synchronization on a different connection:

- Execute a SELECT CONNECTION_PROPERTY statement to determine the connection ID of the current connection.
- Execute a SYNCHRONIZE statement to start synchronization.
- On a separate connection, use the sp_get_last_synchronize_results system procedure to retrieve results using the connection ID you determined above.

To view the results of a synchronization that is complete or in progress on a specific connection, you can use the sp_get_last_synchronize_results system procedure.

The SYNCHRONIZE statement is similar to the UltraLite SYNCHRONIZE statement. However, the SQL Anywhere SYNCHRONIZE statement launches the dbmlsync utility in server mode to perform the synchronization. The UltraLite SYNCHRONIZE statement uses UltraLite runtime.

The database server functions as a dbmlsync API client and uses TCP/IP to communicate with a dbmlsync server. By default, this communication occurs on port 4433. Use the PORT clause to specify a different port.

Use the SYNCHRONIZE PROFILE and SYNCHRONIZE USING statements to perform a synchronization. Use the SYNCHRONIZE START and SYNCHRONIZE STOP statements to start or stop a dbmlsync server. When executing a SYNCHRONIZE PROFILE or SYNCHRONIZE USING statement, the database server attempts to connect to a dbmlsync server that is already running. If a dbmlsync server that is already running cannot be located, a dbmlsync server is started. When the synchronization is complete, the database server shuts down the dbmlsync server it started. If the statement connected to a dbmlsync server that was already running, the dbmlsync server is not shut down. If you are performing multiple synchronizations and do not want to start and stop the dbmlsync server for each synchronization, you can execute a SYNCHRONIZE START statement, followed by multiple SYNCHRONIZE PROFILE or SYNCHRONIZE USING statements, and end with a SYNCHRONIZE STOP statement.

**Privileges**

You must have either the MANAGE REPLICATION system privilege or the SYS_RUN_REPLICATION_ROLE system role.
Side effects

None

See also

● “CREATE SYNCHRONIZATION PROFILE statement [MobiLink]” on page 685
● “Events and parm_id values from the synchronize_parameters table” on page 1282
● “DBSC_Event structure [Dbmlsync .NET]” [MobiLink - Client Administration]
● “MobiLink synchronization” [MobiLink - Getting Started]
● “MobiLink synchronization profiles” [MobiLink - Client Administration]
● “SQL Anywhere MobiLink client deployment” [MobiLink - Server Administration]
● “sp_get_last_synchronize_result system procedure” on page 1281

Standards and compatibility

● SQL/2008  Vendor extension.

Example

The following example shows the syntax for synchronizing a synchronization profile named Test1:

    SYNCHRONIZE PROFILE Test1;

STOP SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]

Stops a MobiLink synchronization schema change.

Syntax

    STOP SYNCHRONIZATION SCHEMA CHANGE

Remarks

The STOP SYNCHRONIZATION SCHEMA CHANGE statement stops a schema change started by a START SYNCHRONIZATION SCHEMA CHANGE statement. All locks obtained by the START SYNCHRONIZATION SCHEMA CHANGE statement are released.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

None.

See also

● “START SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]” on page 1002
● “SYSARTICLE system view” on page 1348
● “SynchronizationSchemaChangeActive database property” [SQL Anywhere Server - Database Administration]
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION SCHEMA CHANGE and STOP SYNCHRONIZATION SCHEMA CHANGE:

```
START SYNCHRONIZATION SCHEMA CHANGE
   FOR TABLES GROUPO.SalesOrders, GROUPO.Products
   SET SCRIPT VERSION = 'version_2' ON SUBSCRIPTION sub1,
   SET SCRIPT VERSION = 'version_2' ON SUBSCRIPTION sub2;
ALTER TABLE GROUPO.SalesOrders ADD SUBTOTAL NUMERIC (10,2);
ALTER TABLE GROUPO.Products ALTER QUANTITY BIGINT;
STOP SYNCHRONIZATION SCHEMA CHANGE;
```

SYNCHRONIZE SUBSCRIPTION statement [SQL Remote]

Synchronizes a subscription for a user to a publication.

Syntax

```
SYNCHRONIZE SUBSCRIPTION
   TO publication-name [(subscription-value)]
   FOR remote-user, ...
```

Parameters

- **publication-name**  The name of the publication to which the user is being subscribed. This may include the owner of the publication.

- **subscription-value**  A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.

- **remote-user**  The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Remarks

A SQL Remote subscription is synchronized when the data in the remote database is consistent with that in the consolidated database, so that publication updates sent from the consolidated database to the remote database will not result in conflicts and errors.

To synchronize a subscription, a copy of the data in the publication at the consolidated database is sent to the remote database. The SYNCHRONIZE SUBSCRIPTION statement does this through the message system. It is recommended that where possible you use the database extraction utility (dbxtract) instead to synchronize subscriptions without using a message system.

Privileges

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

Side effects

Automatic commit.
See also

- “CREATE SUBSCRIPTION statement [SQL Remote]” on page 683
- “START SUBSCRIPTION statement [SQL Remote]” on page 1000
- “STOP SUBSCRIPTION statement [SQL Remote]” on page 1008
- “Extraction utility (dbxtract)” [SQL Remote]
- “Synchronizing with the SQL Remote Message Agent (dbremote)” [SQL Remote]
- “Subscription resynchronization” [SQL Remote]
- “Synchronizing subscriptions” [SQL Remote]

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following statement synchronizes the subscription of user Sam_Singer to the pub_contact publication.

```
SYNCHRONIZE SUBSCRIPTION
TO pub_contact
FOR Sam_Singer;
```

**SYSTEM statement [Interactive SQL]**

Launches an executable file from within Interactive SQL.

**Syntax**

```
SYSTEM [ path ] filename
```

**Remarks**

Launches the specified executable file. The path and file name must be enclosed in single quotation marks.

**Privileges**

None.

**Side effects**

None.

**See also**

- “CONNECT statement [ESQL] [Interactive SQL]” on page 543
- “Interactive SQL” [SQL Anywhere Server - Database Administration]
- “Interactive SQL utility (dbisql)” [SQL Anywhere Server - Database Administration]

**Standards and compatibility**

- SQL/2008  Vendor extension.
Example

The following statement launches the Notepad program if the Notepad executable is in your path.

```sql
SYSTEM 'notepad.exe';
```

**TRIGGER EVENT statement**

Triggers a named event. The event may be defined for event triggers or be a scheduled event.

**Syntax**

```
TRIGGER EVENT event-name [ ( parm = value, ... ) ]
```

**Parameters**

- `parm = value` When a triggering condition causes an event handler to execute, the database server can provide context information to the event handler using the `event_parameter` function. The `TRIGGER EVENT` statement allows you to explicitly supply these parameters, to simulate a context for the event handler.

**Remarks**

Actions are tied to particular trigger conditions or schedules by a `CREATE EVENT` statement. You can use the `TRIGGER EVENT` statement to force the event handler to execute, even when the scheduled time or trigger condition has not occurred. `TRIGGER EVENT` does not execute disabled event handlers.

Each `value` is a string. The maximum length of each `value` is limited by the maximum page size specified by the `-gp` server option. If the length of `value` exceeds the page size, the string is truncated at the point at which the page is full.

**Privileges**

You must be the owner of the event, or have the MANAGE ANY EVENT system privilege.

**Side effects**

None

**See also**

- "-gp database server option" [SQL Anywhere Server - Database Administration]
- “ALTER EVENT statement” on page 436
- “CREATE EVENT statement” on page 570
- “EVENT_PARAMETER function [System]” on page 240

**Standards and compatibility**

- **SQL/2008** Vendor extension.

**Example**

The following example shows how to pass a string parameter to an event. The event displays the time it was triggered in the database server messages window.
CREATE EVENT ev_PassedParameter
HANDLER
BEGIN
   MESSAGE 'ev_PassedParameter - was triggered at ' ||
   event_parameter( 'time' );
END;
TRIGGER EVENT ev_PassedParameter( "Time"=string( current timestamp ) );

TRUNCATE statement

Deletes all rows from a table without deleting the table definition.

Syntax

TRUNCATE
TABLE [ owner.]{table-name}
| MATERIALIZED VIEW [ owner.]{materialized-view-name}

Remarks

The TRUNCATE statement deletes all rows from the table or materialized view.

Note

The TRUNCATE TABLE statement should be used with great care on a database involved in
synchronization or replication because the statement deletes all rows from a table, similar to a DELETE
statement that doesn't have a WHERE clause. However, no triggers are fired as a result of a TRUNCATE
statement. Furthermore, the row deletions are not entered into the transaction log and therefore are not
synchronized or replicated. This can lead to inconsistencies that can cause synchronization or replication
to fail.

After a TRUNCATE statement, the object's schema and all the indexes continue to exist until you execute
a DROP statement. The schema definitions and constraints remain intact, and triggers and privileges
remain in effect.

table-name can be the name of a base table or a temporary table.

With TRUNCATE TABLE, if all the following criteria are satisfied, a fast form of table truncation is
executed:

● There are no foreign keys either to or from the table.

● The TRUNCATE TABLE statement is not executed within a trigger.

● The TRUNCATE TABLE statement is not executed within an atomic statement.

If a fast truncation is carried out, individual DELETEs are not recorded in the transaction log, and a
COMMIT is carried out before and after the operation. Fast truncation cannot be used within snapshot
transactions.

If you attempt to use TRUNCATE TABLE on a table on which an immediate text index is built, or that is
referenced by an immediate view, the truncation fails. This does not occur for non-immediate text indexes
or materialized views; however, it is strongly recommended that you truncate the data in dependent
indexes and materialized views before executing the TRUNCATE TABLE statement on a table, and then refreshing the indexes and materialized views after.

For base tables and materialized views, the TRUNCATE statement requires exclusive access to the table, as the operation is atomic (either all rows are deleted, or none are). Any cursors that were previously opened and that reference the table being truncated must be closed and a COMMIT or ROLLBACK must be executed to release the reference to the table.

For temporary tables, each user has their own copy of the data, and exclusive access is not required when executing the TRUNCATE statement.

**Privileges**

To execute this statement, one of the following conditions must be true:

- You must be the owner of the table.
- You have ALTER privilege on the table
- You have TRUNCATE privilege on the table
- You have the ALTER ANY TABLE system privilege
- You have the TRUNCATE ANY TABLE system privilege
- You have the ALTER ANY OBJECT system privilege

**Side effects**

- When you truncate a materialized view, you change the status of the view to uninitialized.
- Delete triggers are not fired by the TRUNCATE statement.
- A COMMIT is performed before and after a TRUNCATE statement is executed.
- Individual deletions of rows are not entered into the transaction log, so the TRUNCATE operation is not replicated. Do not use this statement in SQL Remote replication or on a MobiLink remote database.
- If the table contains a column defined as DEFAULT AUTOINCREMENT or DEFAULT GLOBAL AUTOINCREMENT, the truncation operation resets the next available value for the column.

**See also**

- “DELETE statement” on page 741
- “TRUNCATE TEXT INDEX statement” on page 1021
- “Deletion of all rows from a table” [SQL Anywhere Server - SQL Usage]
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]
- “Advanced: Status and properties for materialized views” [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**

- **SQL/2008** The TRUNCATE TABLE statement is optional language feature F200 of the SQL/2008 standard. TRUNCATE MATERIALIZED VIEW is a vendor extension.

**Example**

Delete all rows from the SalesOrderItems table:

```sql
TRUNCATE TABLE GROUPO.SalesOrderItems;
```
**TRUNCATE TEXT INDEX statement**

Deletes the data in a MANUAL or an AUTO REFRESH text index.

**Syntax**

```sql
TRUNCATE TEXT INDEX text-index-name
ON [ owner.]table-name
```

**Parameters**

- **ON clause**  The name of the table on which the text index is built.

**Remarks**

Use the TRUNCATE TEXT INDEX statement when you want to delete data from a manual text index without dropping the text index definition. For example, to alter the text configuration object for the text index to change the stoplist, truncate the text index, change the text configuration object it refers to, and then refresh the text index to populate it with new data.

You cannot perform a TRUNCATE TEXT INDEX statement on a text index defined as IMMEDIATE REFRESH (the default). For IMMEDIATE REFRESH text indexes, you must drop the index instead.

The TRUNCATE TEXT INDEX requires exclusive access to the table. Any open cursors that reference the table being truncated must be closed, and a COMMIT or ROLLBACK statement must be executed to release the reference to the table.

**Privileges**

You must be the owner of the table, or have one of the following privileges:

- ALTER privilege on the table
- ALTER ANY INDEX system privilege
- ALTER ANY TABLE system privilege

**Side effects**

Automatic commit

**See also**

- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Text index concepts and reference” [SQL Anywhere Server - SQL Usage]
- “CREATE TEXT INDEX statement” on page 711
- “ALTER TEXT INDEX statement” on page 504
- “DROP TEXT INDEX statement” on page 782
- “REFRESH TEXT INDEX statement” on page 928

**Standards and compatibility**

- SQL/2008  Vendor extension.
Example

The first statement creates the txt_index_manual text index. The second statement populates the text index with data. The third statement truncates the text index data.

```
CREATE TEXT INDEX txt_index_manual ON GROUPO.MarkingInformation
    ( Description )
    MANUAL REFRESH;
REFRESH TEXT INDEX txt_index_manual ON GROUPO.MarkingInformation;
TRUNCATE TEXT INDEX txt_index_manual ON GROUPO.MarkingInformation;
```

The truncated text index is repopulated with data the next time it is refreshed.

TRY statement

Implements error handling for compound statements (if an error occurs in the TRY block, it passes control to another group of statements that is enclosed in a CATCH block).

Syntax

```
[ statement-label : ]
BEGIN TRY
    [ local-declaration; ... ]
    [ statement-list ]
END TRY
BEGIN CATCH
    [ statement-list ]
END CATCH
```

local-declaration : variable-declaration
| cursor-declaration
| exception-declaration
| temporary-table-declaration

variable-declaration and exception-declaration : See “DECLARE statement” on page 736.

cursor-declaration : See “DECLARE CURSOR statement [ESQL] [SP]” on page 729.

temporary-table-declaration : See “DECLARE LOCAL TEMPORARY TABLE statement” on page 735.

Parameters

**statement-label**  When an ending *statement-label* is specified, it must match the beginning *statement-label*. The LEAVE statement can be used to resume execution at the first statement after the compound statement. The compound statement that is the body of a procedure or trigger has an implicit label that is the same as the name of the procedure or trigger.

**local-declaration**  Immediately following the BEGIN TRY, a compound statement can have local declarations for objects that only exist within the compound statement. A compound statement can have a local declaration for a variable, a cursor, a temporary table, or an exception. Local declarations can be referenced by any statement in that compound statement, or in any compound statement nested within it.
Local declarations of the compound statement are visible to the exception handler for the statement. Local declarations are not visible to other procedures that are called from within a compound statement.

Remarks
The CATCH block is the error handler for the TRY statement.

TRY...CATCH statements can be nested and used anywhere that a BEGIN...END statement can be used. Statements within the TRY block ignore the on_tsql_error and continue_after_raiserror database options, as well as the ON EXCEPTION RESUME clause of the CREATE PROCEDURE statement.
TRY...CATCH statements are not atomic.

If no errors occur in the TRY block, the CATCH block is skipped and control passes to the statement following the CATCH block or the caller if no such statement exists. If an error occurs in one of the statements in the TRY block, control passes to the first statement in the CATCH block. Once the CATCH block completes, control passes to the statement following the CATCH block or the caller if no such statement exists. The effect of statements that precede a statement that generates an error is not reverted unless the exception handler generates an error and is nested within an atomic block or an explicit ROLLBACK statement is called. In this case, all statements within the atomic transaction block are reverted.

Errors in the CATCH block are handled according to the connection and procedure settings unless the statements generating them are enclosed in additional TRY...CATCH statements.

Privileges
None.

Side effects
None.
See also

- “ERROR_LINE function [Miscellaneous]” on page 228
- “ERROR_MESSAGE function [Miscellaneous]” on page 229
- “ERROR_PROCEDURE function [function type]” on page 230
- “ERROR_SQLCODE function [Miscellaneous]” on page 231
- “ERROR_SQLSTATE function [Miscellaneous]” on page 232
- “ERROR_STACK_TRACE function [Miscellaneous]” on page 233
- “STACK_TRACE function [Miscellaneous]” on page 371
- “sa_error_stack_trace system procedure” on page 1136
- “sa_stack_trace system procedure” on page 1250
- “Nested compound statements and exception handlers” [SQL Anywhere Server - SQL Usage]
- “Stored procedures, triggers, batches, and user-defined functions” [SQL Anywhere Server - SQL Usage]
- “Error and warning handling” [SQL Anywhere Server - SQL Usage]
- “Exception handling and atomic compound statements” [SQL Anywhere Server - SQL Usage]
- “Exception handlers” [SQL Anywhere Server - SQL Usage]
- “Atomic compound statements” [SQL Anywhere Server - SQL Usage]
- “CONTINUE statement” on page 546
- “SIGNAL statement [SP]” on page 992
- “RESIGNAL statement [SP]” on page 936
- “RAISERROR statement” on page 920
- “BEGIN statement [TSQL]” on page 526

Standards and compatibility

- SQL/2008  Vendor extension.

Example

These examples use the following table:

    CREATE TABLE t( col1 DOUBLE );

Executing the following compound statement inserts value 6 into table t:

    BEGIN TRY
        DECLARE val INT;
        SET val = 0;
        INSERT INTO t VALUES( 1 / val );
        -- This statement will not be executed
        INSERT INTO t VALUES( val );
    END TRY
    BEGIN CATCH
        SET val = 6;
        INSERT INTO t VALUES( val );
    END CATCH;

Executing the following procedure by using CALL proc1(10); inserts the following values into the table t:
CREATE PROCEDURE DBA.proc1( v INTEGER )
BEGIN TRY
    DECLARE local_val INTEGER = 0;
    INSERT INTO t VALUES(-v);
    SET local_val = v / local_val;
    -- This statement will not be executed
    MESSAGE 'The value is ', v;
END TRY
BEGIN CATCH
    INSERT INTO t VALUES(v);
END CATCH;

**UNION statement**

Combines the results of two or more SELECT statements or query expressions.

**Syntax**

```
[ WITH temporary-views ] query-block
UNION [ ALL | DISTINCT ] query-block
[ ORDER BY [ integer | select-list-expression-name ] [ ASC | DESC ], ... ]
[ FOR XML xml-mode ]
[ OPTION( query-hint, ... ) ]
```

query-block : See “Common elements in SQL syntax” on page 421.

query-hint :

**MATERIALIZED VIEW OPTIMIZATION** option-value
| **FORCE OPTIMIZATION**
| option-name = option-value

option-name : identifier

option-value :
hostvar (indicator allowed)
| string
| identifier
| number

**Parameters**

**FOR XML clause**  For information about the FOR XML clause, see “SELECT statement” on page 955.
**OPTION clause**  Use this clause to specify hints for executing the statement. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION *option-value*
- FORCE OPTIMIZATION
- *option-name* = *option-value*. A `OPTION( isolation_level = ... )` specification in the query text overrides all other means of specifying isolation level for a query.

**Remarks**

UNION ALL concatenates the results of the two query blocks into a single (larger) result set. Each query block may be nested. UNION DISTINCT eliminates duplicate rows in the final result. Eliminating duplicates requires extra processing, so UNION ALL should be used instead of UNION where possible. UNION DISTINCT is identical to UNION.

The result sets of the two query blocks must be UNION-compatible; they must each have the same number of items in their respective SELECT lists, and the types of each expression should be comparable. If corresponding items in two SELECT lists have different data types, SQL Anywhere chooses a data type for the corresponding column in the result and automatically convert the columns in each query-block appropriately.

The column names displayed are the same column names that are displayed for the first query-block and these names are used to determine the expression names to be matched with the ORDER BY clause. An alternative way of customizing result set column names is to use a common table expression (the WITH clause).

**Privileges**

You must be the owner of the objects mentioned in query-block, or have SELECT privilege on the objects you are joining in the union.

**Side effects**

None.

**See also**

- “EXCEPT statement” on page 789
- “INTERSECT statement” on page 870
- “OPTION clause, SELECT statement” on page 962
- “SELECT statement” on page 955
- “Set operators and NULL” [SQL Anywhere Server - SQL Usage]

**Standards and compatibility**

- **SQL/2008**  UNION is a core feature of the SQL/2008 standard. Explicitly specifying the DISTINCT keyword with UNION is optional SQL language feature T551. Specifying an ORDER BY clause with UNION is SQL language feature F850. A query-block that contains an ORDER BY clause constitutes SQL/2008 feature F851. A query block that contains a row-limit clause (SELECT TOP or LIMIT) comprises optional SQL language feature F857 or F858, depending on the context. The FOR XML and OPTION clauses are vendor extensions.
**Transact-SQL**  
UNION and UNION ALL are supported by Adaptive Server Enterprise. The FOR XML and OPTION clauses are not supported.

**Example**

List all distinct surnames of employees and customers.

```sql
SELECT Surname
FROM GROUPO.Employees
UNION
SELECT Surname
FROM GROUPO.Customers;
```

**UNLOAD statement**

Unloads data from a data source into a file.

**Syntax**

```
UNLOAD data-source
{ TO filename
 | INTO FILE filename
 | INTO CLIENT FILE client-filename
 | INTO VARIABLE variable-name }
[ unload-option ... ]
```

- `data-source`: `FROM TABLE [owner.]table-name` or `FROM MATERIALIZED VIEW [owner.]materialized-view-name`
- `filename`: string | variable
- `client-filename`: string | variable

**Syntax**

```
unload-option :
| APPEND { ON | OFF }
| BYTE ORDER MARK { ON | OFF }
| { COMPRESSED | NOT COMPRESSED }
| COLUMN DELIMITED BY string
| DELIMITED BY string
| ENCODING encoding
| { ENCRYPTED KEY 'key' [ ALGORITHM 'algorithm' ] | NOT ENCRYPTED }
| ESCAPE CHARACTER character
| ESCAPES { ON | OFF }
| FORMAT { TEXT | BCP }
| HEXADECIMAL { ON | OFF }
| ORDER { ON | OFF }
| QUOTE string
| QUOTES { ON | OFF }
| ROW DELIMITED BY string
```

- `encoding`: string
Parameters

**TO clause**  The name of the file to unload data into. The `filename` path is relative to the database server's starting directory. If the file does not exist, it is created. If it already exists, it is overwritten unless APPEND ON is also specified.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

**INTO FILE clause**  Semantically equivalent to TO `filename`.

If disk sandboxing is enabled, then the database's operations are limited to the directory where the main database file is located. See “Disk sandboxing” [SQL Anywhere Server - Database Administration].

**INTO CLIENT FILE clause**  The file on the client computer into which the data is unloaded. If the file doesn't exist, it is created. If it already exists, it is overwritten unless APPEND ON is also specified. The path is resolved on the client computer relative to the current working directory of the client application.

**INTO VARIABLE clause**  The variable to unload the data into. The variable must already exist and be of CHAR, NCHAR or BINARY type. The APPEND option causes the unloaded data to be concatenated to the current contents of the variable.

**APPEND clause**  When APPEND is ON, unloaded data is appended to the end of the file specified. When APPEND is OFF, unloaded data replaces the contents of the file specified. APPEND is OFF by default. This clause cannot be specified when specifying the COMPRESSED or ENCRYPTED clauses, and cannot be used if the file being appended to is compressed or encrypted.

**BYTE ORDER MARK clause**  Use this clause to specify whether a byte order mark (BOM) should be written. By default, this option is ON, provided the destination for the unload is a local or client file. When the BYTE ORDER MARK option is ON and the ENCODING is UTF-8 or UTF-16, then a BOM is written. If BYTE ORDER MARK is OFF, a BOM is not unloaded.

**COMPRESSED clause**  Specifies whether to compress the data. The default is NOT COMPRESSED. You cannot compress the data if you want the data to be appended (APPEND ON).

**COLUMN DELIMITED BY and DELIMITED BY clauses**  The string used between columns. The default column delimiter is a comma. You can specify an alternative column delimiter by providing a string up to 255 bytes in length.

**ENCODING clause**  All database data is translated from the database character encoding to the specified CHAR or NCHAR encoding. When ENCODING is not specified, the database's CHAR encoding is used.

If a translation error occurs during the unload operation, it is reported based on the setting of the `on_charset_conversion_failure` option.

Specify the BYTE ORDER MARK clause to include a byte order mark in the data.

**ENCRYPTED clause**  Specifies whether to encrypt the data. If you specify NOT ENCRYPTED (the default), the data is not encrypted. If you specify ENCRYPTED KEY with a key and no algorithm, the data is encrypted using AES128 and the specified key. The key can be either a string or a variable name.
If you specify ENCRYPTED KEY with a key and algorithm, the data is encrypted using the specified key and algorithm. The algorithm can be any of the algorithms accepted by the CREATE DATABASE statement. You cannot specify simple encryption.

You cannot encrypt the data if you want the data to be appended (APPEND ON).

**ESCAPES clause**  With ESCAPES turned ON (the default), the database server writes escape sequences. Newline characters can be written as the combination `\n`, other characters can be included in data as hexadecimal ASCII codes, such as `\x09` for the tab character. A sequence of two backslash characters (`\\`) is written as a single backslash. A backslash followed by any character other than n, x, X, or \ is written as two separate characters. For example, `\q` inserts a backslash and the letter q. It is recommended that the string you specify for the escape character is no longer than one multibyte character.

**FORMAT clause**  Outputs data in either TEXT format or in BCP out format. If you choose TEXT, output lines are written as text characters, one row per line, with values separated by the column delimiter string. If you choose BCP, data including BLOBs are exported as BCP input files for use with Adaptive Server Enterprise. The default format is TEXT.

**HEXADECIMAL clause**  By default, HEXADECIMAL is ON. Binary column values are written as `0xnnnnnn...`, where 0x is a zero followed by an x, and each n is a hexadecimal digit. It is important to use HEXADECIMAL ON when dealing with multibyte character sets.

The HEXADECIMAL clause can be used only with the FORMAT TEXT clause.

**ESCAPE CHARACTER clause**  Use this clause to specify the escape character used in the data. The default escape character for characters written as hexadecimal codes and symbols is a backslash (`\`), so `\x0A` is the linefeed character, for example. This can be changed using the ESCAPE CHARACTER clause.

It is recommended that the string you specify for the escape character is no longer than one multibyte character.

**ORDER clause**  With ORDER ON (the default), the exported data is ordered by clustered index if one exists. If a clustered index does not exist, the exported data is ordered by primary key values. With ORDER OFF, the data is exported in the same order you see when selecting from the table without an ORDER BY clause. Exporting is slower with ORDER ON. However, reloading using the LOAD TABLE statement is quicker because of the simplicity of the indexing step.

For UNLOAD select-statement, the ORDER clause is ignored. However, you can still order the data by specifying an ORDER BY clause in the SELECT statement.

**QUOTE clause**  The QUOTE clause is for TEXT data only; the string is placed around string values. The default is a single quote (apostrophe).

**QUOTES clause**  With QUOTES turned on (the default), the quote character, which defaults to a single quote (apostrophe), is placed around all exported strings.

**ROW DELIMITED BY clause**  Use this clause to specify the string that indicates the end of a record. The default delimiter string is a newline (`\n`). However, it can be any string up to up to 255 bytes in
Remarks

The UNLOAD select-statement statement allows data from a SELECT statement to be exported to a comma-delimited file. The result set is not ordered unless the SELECT statement contains an ORDER BY clause.

The UNLOAD TABLE statement allows efficient mass exporting from a database table or materialized view into a file. The UNLOAD TABLE statement is more efficient than the Interactive SQL statement OUTPUT, and can be called from any client application.

The database server, or the client application, depending upon whether INTO FILE or INTO CLIENT FILE was specified, respectively, must have operating system permissions to write to the specified file.

When unloading table columns with binary data types, UNLOAD TABLE writes hexadecimal strings, of the form \xnnnn, where n is a hexadecimal digit.

When unloading and reloading a database that has proxy tables, you must create an external login to map the local user to the remote user, even if the user has the same password on both the local and remote databases. If you do not have an external login, the reload may fail because you cannot connect to the remote server.

When unloading into a variable (INTO VARIABLE), the output is converted to a character set as follows:

- **CHAR** write to the variable in CHAR encoding. The ENCODING clause must match the CHAR encoding.

- **NCHAR** write to the variable in NCHAR encoding. The ENCODING clause must match the NCHAR encoding.

- **BINARY** write to the variable in BINARY encoding. The ENCODING clause must match the BINARY encoding; otherwise, the CHAR encoding is used.

If you choose to compress and encrypt the unloaded data, it is compressed first.

UNLOAD TABLE places an exclusive lock on the whole table or materialized view.

During the execution of this statement, you can request progress messages.

You can also use the Progress connection property to determine how much of the statement has been executed.

To retain maximum precision of date values, set the date_format to YYYY-MM-DD.

To retain maximum precision of TIMESTAMP values, set the timestamp_format to YYYY-MM-DD HH:NN:SS.SSSSSS.

To retain maximum precision of TIMESTAMPM WITH TIMEZONE values, set the timestamp_with_time_zone_format to YYYY-MM-DD HH:NN:SS.SSSSSS+HH:NN.
Privileges

When unloading into a variable, no privileges are required. Otherwise, the required privileges depend on the database server -gl option, as follows:

- If the -gl option is set to ALL, you must be the owner of the tables, or have SELECT privilege on the tables, or have the SELECT ANY TABLE system privilege.
- If the -gl option is set to DBA, you must have the SELECT ANY TABLE system privilege.
- If the -gl option is set to NONE, UNLOAD is not permitted.

When unloading to a file on a client computer:

- You must have the WRITE CLIENT FILE privilege.
- You must have write permissions on the directory where the file is located.
- The allow_write_client_file database option must be enabled.
- The write_client_file secure feature must be enabled.

Side effects

None. The query is executed at the current isolation level.

See also

- “CREATE DATABASE statement” on page 549
- “LOAD TABLE statement” on page 873
- “PASSTHROUGH statement [SQL Remote]” on page 913
- “Clustered indexes” [SQL Anywhere Server - SQL Usage]
- “Supported character sets” [SQL Anywhere Server - Database Administration]
- “OUTPUT statement [Interactive SQL]” on page 907
- “Tips on exporting data with the UNLOAD statement” [SQL Anywhere Server - SQL Usage]
- “Access to data on client computers” [SQL Anywhere Server - SQL Usage]
- “Data import and export” [SQL Anywhere Server - SQL Usage]
- “Tips on exporting data with the UNLOAD TABLE statement” [SQL Anywhere Server - SQL Usage]
- “-sf database server option” [SQL Anywhere Server - Database Administration]
- “allow_write_client_file option” [SQL Anywhere Server - Database Administration]
- “-gl database server option” [SQL Anywhere Server - Database Administration]
- “timestamp_format option” [SQL Anywhere Server - Database Administration]
- “date_format option” [SQL Anywhere Server - Database Administration]
- “on_charset_conversion_failure option” [SQL Anywhere Server - Database Administration]
- “Progress connection property” [SQL Anywhere Server - Database Administration]
- “progress_messages option” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following example unloads the contents of the Products table to a UTF-8-encoded file, productsT.dat:
UNLOAD TABLE GROUPO.Products TO 'c:\temp\productsT.dat' ENCODING 'UTF-8';

The following example creates a variable called @myProducts and then unloads the Products.Name column into the variable:

```sql
CREATE VARIABLE @myProducts LONG VARCHAR;
UNLOAD SELECT NAME FROM GROUPO.Products INTO VARIABLE @myProducts ESCAPE CHARACTER '!!';
```

### UPDATE (positioned) statement [ESQL] [SP]

Modifies the data at the current location of a cursor.

**Syntax 1** [ESQL only]

```
UPDATE WHERE CURRENT OF cursor-name
{ USING [ SQL ] DESCRIPTOR sqlda-name | [ FROM ] | [ USING ] } hostvar-list
```

**Syntax 2**

```
UPDATE update-table, ...
SET set-item, ...
WHERE CURRENT OF cursor-name

hostvar-list : indicator variables allowed

update-table :
[owner-name.]object-name [ [ AS ] correlation-name ]

set-item :
[ correlation-name.]column-name = { expression | DEFAULT }
[[ owner-name.]object-name.column-name = { expression | DEFAULT }

object-name : identifier (a table or view name)

sqlda-name : identifier
```

**Parameters**

**UPDATE clause**  
Each `update-table` is matched to a table in the query for the cursor as follows:

1. If a correlation name is specified, it is matched to a table in the cursor's query that has the same `table-or-view-name` and the same `correlation-name`.

2. If there is a table in the cursor's query that has the same `table-or-view-name` that does not have a correlation name or has a correlation name that is the same as the `table-or-view-name`, then the update table is matched with this table in the cursor's query.

3. If there is a single table in the cursor's query that has the same `table-or-view-name` as the update table, then the update table is matched with this table in the cursor's query.

If a column has a default defined, use the SET clause to set a column to its default value. For an example, see the Examples section of “UPDATE statement” on page 1037.
**USING DESCRIPTOR clause**  When assigning a variable, the variable must already be declared, and its name must begin with the at sign (@). Variable and column assignments can be mixed together, and any number can be used.

**SET clause**  The columns in set-item must be in the table or view being updated. If a name on the left side of an assignment in the SET list matches a column in the updated table and the variable name, the statement updates the column. set-item cannot refer to aliases or columns from other tables or views. If the table or view you are updating is given a correlation name in the cursor specification, use the correlation name in the SET clause.

Each set-item is associated with a single update-table, and the corresponding column of the matching table in the cursor's query is modified. expression references columns of the tables identified in the UPDATE list and can use constants, host variables, variables, expressions from the SELECT list of the query, or combinations of the above using operators such as +, -, ..., COALESCE, IF, and so on. expression cannot reference aliases of expressions from the cursor's query or columns of other tables of the cursor's query that are not in the UPDATE list. Subselects, subquery predicates, and aggregate functions cannot be used in set-item.

**Remarks**

This form of the UPDATE statement updates the current row of the specified cursor. The current row is defined to be the last row successfully fetched from the cursor, and the last operation on the cursor must not have been a positioned DELETE statement.

For syntax 1, columns from the SQLDA or values from the host variable list correspond one-to-one with the columns returned from the specified cursor. If the sqldata pointer in the SQLDA is the null pointer, the corresponding SELECT list item is not updated.

In syntax 2, the requested columns are set to the specified values for the row at the current row of the specified query. The columns do not need to be in the SELECT list of the specified open cursor. This format can be prepared.

Also, when assigning a variable, the variable must already be declared, and its name must begin with the at sign (@). Variable and column assignments can be mixed together, and any number can be used. If a name on the left side of an assignment in the SET list matches a column in the updated table and the variable name, the statement updates the column.

The USING DESCRIPTOR, FROM hostvar-list, and hostvar formats are for embedded SQL only.

**Privileges**

You must be the owner of the table being updated, or have UPDATE privilege on the columns being modified, or have the UPDATE ANY TABLE system privilege.

**Side effects**

None.
Standards and compatibility

- **SQL/2008**  Syntax 1 is a vendor extension. Syntax 2 is a core feature of the SQL/2008 standard. If used within an embedded SQL program, Syntax 2 comprises part of optional SQL language feature B031, "Basic dynamic SQL". The ability to specify more than one table to be updated is a vendor extension.

  The range of cursors that can be updated is dependent upon the setting of the ansi_update_constraints option. The ability to perform a positioned update over a cursor that is ordered—that is the SQL query has an ORDER BY clause—comprises optional SQL/2008 language feature F831, "Full cursor update". Performing a positioned update over more complex SQL constructions may involve additional vendor extensions.

Example

The following is an example of an UPDATE statement executed on a fictitious cursor called emp_cursor:

```
UPDATE GROUPO.Employees
SET Surname = 'Jones'
WHERE CURRENT OF emp_cursor;
```

**UPDATE statement [SQL Remote]**

Modifies data in the database.

**Syntax 1**

```
UPDATE table-list
SET column-name = expression, ...
[ VERIFY ( column-name, ... ) VALUES ( expression, ... ) ]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ], ... ]
```

**Syntax 2**

```
UPDATE table-name
PUBLICATION publication-name
{ SUBSCRIBE BY subscription-expression |
  OLD SUBSCRIBE BY old-subscription-expression
  NEW SUBSCRIBE BY new-subscription-expression }
WHERE search-condition

expression : value | subquery
```
Parameters

- **table-name** The table-name indicates the table that must be modified on the remote databases.

- **publication-name** The publication-name indicates the publication for which subscriptions must be changed.

- **subscription-expression** The value of subscription-expression is used by SQL Remote to determine both new and existing recipients of the rows. The subscription-expression is either a value or a subquery. Alternatively, you can provide both OLD and NEW subscription expressions.

- **WHERE clause** The WHERE clause specifies which rows are to be transferred between subscribed databases.

Remarks

The UPDATE statement is used to modify rows of one or more tables. Each named column is set to the value of the expression on the right side of the equal sign. There are no restrictions on the expression. Even column-name can be used in the expression—the old value is used.

If no WHERE clause is specified, every row is updated. If a WHERE clause is specified, then only those rows which satisfy the search condition are updated.

Normally, the order that rows are updated does not matter. However, with the NUMBER(*) function, an ordering can be useful to get increasing numbers added to the rows in some specified order. Also, to do something like add 1 to the primary key values of a table, it is necessary to do this in descending order by primary key, so that you do not get duplicate primary keys during the operation.

Views can be updated provided the SELECT statement defining the view does not contain a GROUP BY clause, an aggregate function, or involve a UNION clause.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case sensitive or not. So, a character data type column updated with a string Value is always held in the database with an uppercase V and the remainder of the letters lowercase. SELECT statements return the string as Value. If the database is not case sensitive, however, all comparisons make Value the same as value, VALUE, and so on. Further, if a single-column primary key already contains an entry Value, an INSERT of value is rejected, as it would make the primary key not unique.

The optional FROM clause allows tables to be updated based on joins. If the FROM clause is present, the WHERE clause qualifies the rows of the FROM clause. Data is updated only in the table list immediately following the UPDATE keyword.

If a FROM clause is used, it is important to qualify the table name that is being updated the same way in both parts of the statement. If a correlation name is used in one place, the same correlation name must be used in the other. Otherwise, an error is generated.

Syntax 1 and Syntax 2 are applicable only to SQL Remote.

Syntax 2 with no OLD and NEW SUBSCRIBE BY expressions must be used in a BEFORE trigger.

Syntax 2 with OLD and NEW SUBSCRIBE BY expressions can be used anywhere.
Syntax 1 is intended for use with SQL Remote only, in single-row updates executed by the Message Agent. The VERIFY clause contains a set of values that are expected to be present in the row being updated. If the values do not match, any RESOLVE UPDATE triggers are fired before the UPDATE proceeds. The UPDATE does not fail if the VERIFY clause fails to match. When the VERIFY clause is specified, only one table can be updated at a time.

Syntax 2 is intended for use with SQL Remote only. If no OLD and NEW expressions are used, it must be used inside a BEFORE trigger so that it has access to the relevant values. The purpose is to provide a full list of subscribe by values any time the list changes. It is placed in SQL Remote triggers so that the database server can compute the current list of SUBSCRIBE BY values. Both lists are placed in the transaction log.

The Message Agent uses the two lists to make sure that the row moves to any remote database that did not have the row and now needs it. The Message Agent also removes the row from any remote database that has the row and no longer needs it. A remote database that has the row and still needs it is not affected by the UPDATE statement.

Syntax 2 of the UPDATE statement allows the old SUBSCRIBE BY list and the new SUBSCRIBE BY list to be explicitly specified, which can make SQL Remote triggers more efficient. In the absence of these lists, the database server computes the old SUBSCRIBE BY list from the publication definition. Since the new SUBSCRIBE BY list is commonly only slightly different from the old SUBSCRIBE BY list, the work to compute the old list may be done twice. By specifying both the old and new lists, this extra work can be avoided.

The OLD and NEW SUBSCRIBE BY syntax is especially useful when many tables are being updated in the same trigger with the same subscribe by expressions. This can dramatically increase performance.

The SUBSCRIBE BY expression is either a value or a subquery.

Syntax 2 of the UPDATE statement is used to implement a specific SQL Remote feature, and is to be used inside a BEFORE trigger.

For publications created using a subquery in a subscription expression, you must write a trigger containing syntax 2 of the UPDATE statement to ensure that the rows are kept in their proper subscriptions.

Syntax 2 of the UPDATE statement makes an entry in the transaction log, but does not change the database table.

**Privileges**

You must be the owner of the table being updated, or have UPDATE privilege on the columns being modified, or have the UPDATE ANY TABLE system privilege.

**Side effects**

None.

**See also**

- “BEFORE UPDATE triggers” [SQL Remote]
Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example transfers employee Philip Chin (employee 129) from the sales department to the marketing department.

```
UPDATE GROUPO.Employees
SET DepartmentID = 400
WHERE EmployeeID = 129;
```

**UPDATE statement**

Modifies existing rows in database tables.

**Syntax 1**

```
UPDATE [ row-limitation ] table-expression [, ...]
SET set-item[, ...]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ] , ...]
[ OPTION( query-hint, ... ) ]
```

*table-expression*: A table expression that can include joins, outer joins, views and derived tables. See “FROM clause” on page 810.

**Syntax 2**

```
UPDATE [ row-limitation ] table-name
SET set-item[, ...]
FROM table-expression [, ...]
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ] , ...]
[ OPTION( query-hint, ... ) ]
```

*table-name*:

- [ owner.]table-name [ [ AS ] correlation-name ]
- [ owner.]view-name [ [ AS ] correlation-name ]
- derived-table

*derived-table* :

- ( select-statement )
- [ AS ] correlation-name [ ( column-name [, ...] ) ]

**Syntax 3**

```
UPDATE table-name
SET set-item, ...
VERIFY ( column-name, ... ) VALUES ( expression, ... )
[ WHERE search-condition ]
[ ORDER BY expression [ ASC | DESC ] , ...]
[ OPTION( query-hint, ... ) ]
```
Syntax 4

```sql
UPDATE [ owner.]table-name
PUBLICATION publication
{ SUBSCRIBE BY expression
| OLD SUBSCRIBE BY expression NEW SUBSCRIBE BY expression }
WHERE search-condition

row-limitation:
FIRST
| TOP { ALL | limit-expression } [ START AT startat-expression ]

limit-expression: simple-expression
startat-expression: simple-expression

simple-expression:
integer
| variable
| ( simple-expression )
| ( simple-expression { + | - | * } simple-expression )

set-item:
[ correlation-name.column-name={ expression | DEFAULT }
| [ owner-name.]table-name.column-name = { expression | DEFAULT }
| @variable-name=expression

query-hint:
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| FORCE NO OPTIMIZATION
| option-name = option-value

table-name:
[ owner.]base-table-name
| temporary-table-name
| derived-table-name
| [ owner.]view-name

option-name: identifier

option-value:
hostvar (indicator allowed)
| string
| identifier
| number
```

Parameters

**UPDATE clause**  For Syntax 1 and Syntax 2, `table-expression` can include temporary tables, global temporary tables, derived tables, or views. Views and derived tables can be updated unless they are non-updatable. For Syntax 1, a list of more than one `table-expression` results in a Cartesian product of the rows formed by the underlying table expressions, which can then be restricted via the use of the WHERE clause. Both Syntax 1 and Syntax 2 permit the updating of joins. For Syntax 3 and Syntax 4, `table-name` must be a base table.

UPDATES can be performed on views only if the query specification defining the view is updatable.
**row-limitation clause**  The row limitation clause allows you to restrict the rows being updated to only a subset of the rows that satisfy the WHERE clause. The TOP and START AT arguments can be simple arithmetic expressions over host variables, integer constants, or integer variables. The TOP argument must evaluate to a value greater than or equal to 0. The START AT argument must evaluate to a value greater than 0. An ORDER BY clause should be used to order the rows in a meaningful manner.

**SET clause**  Use the SET clause to set column names or variables to the specified expression.

You can use the SET clause to set the column to a computed column value using this format:

```
SET column-name = expression, ...
```

Each specified column is set to the value of the expression. There are no restrictions on expression. If expression is a column-name, the previous value from that column is used.

If a column has a default defined, use the SET clause to set a column to its default value.

You can also use the SET clause to assign a variable by using the following format:

```
SET @variable-name = expression, ...
```

When assigning a value to a variable, the variable must already be declared, and its name must begin with the at sign (@). If the variable name matches the name of a column in the table to be updated, the UPDATE statement updates the column value and leaves the variable unchanged. Variable and column assignments can be combined in any order.

**FROM clause**  The FROM table-expression clause allows tables to be updated based on joins. table-expression can contain arbitrary complex table expressions, such as OUTER, CROSS, and NATURAL joins.

If the FROM clause is present, table-name must specify the sole table to be updated, and it must qualify the name in the same way as it appears in the FROM clause. If correlation names are used in the FROM clause, the identical correlation name must be specified as table-name. If the table expression to be updated is a derived table, the derived table must be repeated in the table-name specification.

Syntax 2 cannot be used if the ansi_update_constraints option is set to Strict.

If a FROM clause is specified, the SET clause can specify only columns from table-name to be updated. Otherwise, an error is generated.

The following statement illustrates a potential ambiguity in table names in UPDATE statements using Syntax 2 that contain table expressions that use correlation names:

```
UPDATE table_1
SET column_1 = ...
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

In the following example, each instance of table_1 in the FROM clause has a correlation name, denoting a self-join of table_1 to itself. However, the UPDATE statement fails to specify which of the rows that make up the self-join are to be updated. This omission can be corrected by specifying the correlation name in the UPDATE statement as follows:
UPDATE table_1 as alias_1
SET column_1 = ...
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...

WHERE clause  If a WHERE clause is specified, only rows satisfying the search condition are updated. If no WHERE clause is specified, every row is updated.

ORDER BY clause  Normally, the order in which rows are updated does not matter. However, with the FIRST or TOP clause, the order can be significant.

You cannot use ordinal column numbers in the ORDER BY clause.

To use the ORDER BY clause, the ansi_update_constraints option must not be set to Strict.

To update columns that appear in the ORDER BY clause, the ansi_update_constraints option must be set to Off.

OPTION clause  Use this clause to specify hints for executing the statement. The following hints are supported:

- MATERIALIZED VIEW OPTIMIZATION option-value
- FORCE OPTIMIZATION
- FORCE NO OPTIMIZATION
- option-name = option-value. A OPTION( isolation_level = ... ) specification in the query text overrides all other means of specifying isolation level for a query.

Remarks

Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive or not. A CHAR data type column updated with the string Street is always held in the database with an uppercase S and the remainder of the letters lowercase. SELECT statements return the string as Street. If the database is not case sensitive, however, all comparisons make Street the same as street, STREET, and so on. Further, if a single-column primary key already contains an entry Street, an UPDATE of another row's primary key to street is rejected, as it would make the primary key not unique.

If the new value does not differ from the old value, no change is made to the data. However, BEFORE UPDATE triggers fire any time an UPDATE occurs on a row, whether the new value differs from the old value. AFTER UPDATE triggers fire only if the new value is different from the old value.

Syntax 3 and 4 are applicable only to SQL Remote.

Syntax 3 is intended for use with SQL Remote only, in single-row updates of a single table executed by the Message Agent. The VERIFY clause contains a set of values that are expected to be present in the row being updated. If the values do not match, any RESOLVE UPDATE triggers are fired before the UPDATE proceeds. The UPDATE does not fail simply because the VERIFY clause fails to match.

Syntax 4 of the UPDATE statement is used to implement a specific SQL Remote feature, and is to be used inside a BEFORE trigger. It provides a full list of SUBSCRIBE BY values any time the list changes. It is placed in SQL Remote triggers so that the database server can compute the current list of SUBSCRIBE BY values. Both lists are placed in the transaction log.
The Message Agent uses the two lists to make sure that the row moves to any remote database that did not have the row and now needs it. The Message Agent also removes the row from any remote database that has the row and no longer needs it. A remote database that has the row and still needs it is not affected by the UPDATE statement.

For publications created using a subquery in a SUBSCRIBE BY clause, you must write a trigger containing syntax 4 of the UPDATE statement to ensure that the rows are kept in their proper subscriptions.

Syntax 4 of the UPDATE statement allows the old SUBSCRIBE BY list and the new SUBSCRIBE BY list to be explicitly specified, which can make SQL Remote triggers more efficient. In the absence of these lists, the database server computes the old SUBSCRIBE BY list from the publication definition. Since the new SUBSCRIBE BY list is commonly only slightly different from the old SUBSCRIBE BY list, the work to compute the old list may be done twice. By specifying both the old and new lists, you can avoid this extra work.

The SUBSCRIBE BY expression is either a value or a subquery.

Syntax 4 of the UPDATE statement makes an entry in the transaction log, but does not change the database table.

Updating a significant amount of data using the UPDATE statement also updates column statistics.

**Privileges**

You must be the owner of the table being updated, or have UPDATE privilege on the columns being modified, or have the UPDATE ANY TABLE system privilege.

**Side effects**

Column statistics are updated to reflect the modified values.

If a table has a primary key, a UNIQUE constraint, or a UNIQUE index, the processing of the UPDATE statement may involve the use of a temporary table if the table manipulations cannot be performed without violating the uniqueness constraint. The temporary table stores rows modified by the UPDATE statement that violate one or more uniqueness constraints. These rows are temporarily deleted from the base table during the execution of the UPDATE statement, and are subsequently re-inserted. This behavior may have implications for AFTER triggers and other concurrent connections.
See also

- “DELETE statement” on page 741
- “DELETE statement (positioned) [ESQL] [SP]” on page 739
- “INSERT statement” on page 860
- “UPDATE statement [SQL Remote]” on page 1034
- “CREATE TRIGGER statement” on page 713
- “CREATE TABLE statement” on page 690
- “PUT statement [ESQL]” on page 919
- “OPTION clause, SELECT statement” on page 962
- “MERGE statement” on page 892
- “ALTER TABLE statement” on page 486
- “UPDATE (positioned) statement [ESQL] [SP]” on page 1032
- “UPDATE statements with a VERIFY clause” [SQL Remote]
- “TSEQUAL function [System] (deprecated)” on page 395
- “FROM clause” on page 810
- “Views” [SQL Anywhere Server - SQL Usage]
- “Joins: Retrieving data from several tables” [SQL Anywhere Server - SQL Usage]
- “Locks during updates” [SQL Anywhere Server - SQL Usage]
- “UPDATE statement replication” [SQL Remote]
- “Row limitation clauses in SELECT, UPDATE, and DELETE query blocks” [SQL Anywhere Server - SQL Usage]
- “Data changes using UPDATE” [SQL Anywhere Server - SQL Usage]
- “ansi_update_constraints option” [SQL Anywhere Server - Database Administration]

Standards and compatibility

- **SQL/2008**  Syntax 1 of the UPDATE statement is a core feature of the SQL/2008 standard. Syntax 2 is a vendor extension. Syntax 3 and 4 are vendor extensions for use only with SQL Remote.

Syntax 1 includes support for two optional SQL language features:

- Support for updating a join, possibly including one or more derived tables, comprises part of optional SQL language feature T111, "Updatable joins, unions, and columns".

- Support for modifying a table referenced in a nested subquery that forms part of the search condition for the UPDATE statement comprises optional SQL/2008 language feature F781, "Self-referencing operations".

The following features of Syntax 1 are vendor extensions:

- The FROM and ORDER BY clauses.

- The **row-limitation** clause.

- The ability to specify more than one **table-expression**.

- The ability to update a variable using the SET clause.

- The OPTION clause.
With Syntax 1, the setting of the ansi_update_constraints option controls which forms of table expressions can be modified. To enforce SQL/2008 core feature compatibility, ensure that the ansi_update_constraints option is set to Strict.

Examples

Using the sample database, this example transfers employee Philip Chin (employee 129) from the sales department to the marketing department.

```sql
UPDATE GROUP0.Employees
SET DepartmentID = 400
WHERE EmployeeID = 129;
```

Using the sample database, this example rennumbers all existing sales orders by subtracting 2000 from the ID.

```sql
UPDATE GROUP0.SalesOrders AS orders
SET orders.ID = orders.ID - 2000
ORDER BY orders.ID ASC;
```

This update is possible only if the foreign key of the SalesOrderItems table (referencing the primary key SalesOrders.ID) is defined with the action ON UPDATE CASCADE. The SalesOrderItems table is then updated as well. Because the statement specifies an ORDER BY clause, and the ordering attribute is also specified in the SET clause, the ansi_update_constraints option must be set to Off or an error is returned.

Using the sample database, this example changes the price of a product at isolation level 2, rather than using the current isolation level setting of the database.

```sql
UPDATE GROUP0.Products
SET UnitPrice = 7.00
WHERE ID = 501
OPTION( isolation_level = 2 );
```

This example requires the ansi_update_constraints option to be set to a value other than Strict. Again using the sample database, this example uses Syntax 2 to reset the quantity-on-hand of those Tee Shirts where there exists at least one order whose quantity exceeds the current quantity-on-hand:

```sql
UPDATE GROUP0.Products AS a
SET Quantity = 0
FROM GROUP0.Products a JOIN GROUP0.SalesOrderItems b ON a.ID = b.ProductID
WHERE a.Name = 'Tee Shirt' AND b.Quantity > a.Quantity;
```

This example requires the ansi_update_constraints option to be set to a value other than Strict. In this example, Syntax 1 is used to both reset the quantity-on-hand for those Tee Shirts, and reset the ShipDate for the Tee Shirt order to today's date:

```sql
UPDATE GROUP0.Products a JOIN GROUP0.SalesOrderItems b on a.ID = b.ProductID
SET a.Quantity = 0, b.ShipDate = CAST( NOW() AS DATE)
WHERE a.Name = 'Tee Shirt' AND b.Quantity > a.Quantity
```

This example shows how to update a table to set a column to its default value. In this example, you create a table, MyTable, populate it with data, and then execute an UPDATE statement specifying the SET clause to change the column values to their defaults.

```sql
CREATE TABLE MyTable(
    PK INT PRIMARY KEY DEFAULT AUTOINCREMENT,
```
VALIDATE LDAP SERVER statement

Validates an LDAP server configuration object.

Syntax

VALIDATE LDAP SERVER { ldapua-server-name | ldapua-server-attribs }  
[ CHECK  user-id [ user-dn-string ] ]

Parameters

ldapua-server-name  The name of the LDAP server configuration object to validate. For a full description of this clause, see the CREATE LDAP SERVER statement.

ldapua-server-attribs  When validating an LDAP server configuration object using ldapua-server-attribs, the specified attributes are validated. The URLs are parsed to identify syntax errors. Validation stops and an error is returned if a syntax error occurs.

For a full description of this clause, see the CREATE LDAP SERVER statement.

CHECK clause  Use this clause to specify a user ID to search for on the LDAP server.

Remarks

When a VALIDATE LDAP SERVER statement is executed, a connection to the LDAP server is attempted. If ACCESS ACCOUNT and a password are specified, the values are used to establish the connection to the SEARCH DN URL, validating the SEARCH DN URL, ACCESS ACCOUNT, and ACCESS ACCOUNT password.

When setting up a new server to use LDAP User Authentication, this statement is useful validating changes to an LDAP server configuration object before applying them, and for diagnosing problems between the database server and the LDAP server.

Privileges

You must have the MANAGE ANY LDAP SERVER system privilege.

Side effects

Automatic commit.
See also

- “LDAP user authentication” [SQL Anywhere Server - SQL Usage]
- “CREATE LDAP SERVER statement” on page 603
- “ALTER LDAP SERVER statement” on page 442
- “DROP LDAP SERVER statement” on page 760

Standards and compatibility

- SQL/2008  Vendor extension.

Example

The following example creates and LDAP server configuration object and connects to the LDAP server at hostname voyager, port number 389, using the ACCESS ACCOUNT and password specified in the definition for apps_primary.

```sql
CREATE LDAP SERVER apps_primary2
  SEARCH DN
    URL 'ldap://voyager:389/dc=MyCompany,dc=com??sub?cn=*'
    ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
    IDENTIFIED BY 'Secret99Password'
  AUTHENTICATION URL 'ldap://voyager:389/
  CONNECTION TIMEOUT 3000
  WITH ACTIVATE;
VALIDATE LDAP SERVER apps_primary2;
```

The following example connects to the LDAP server at hostname voyager, port number 389, using the ACCESS ACCOUNT and password specified in the definition for apps_primary2. It also checks that user ID myusername is valid and matches the expected user DN:

```sql
VALIDATE LDAP SERVER apps_primary2
  CHECK myusername 'cn=myusername,cn=Users,dc=mycompany,dc=com';
```

If the LDAP server configuration object has not been defined, the same checks can be performed by specifying the attributes:

```sql
VALIDATE LDAP SERVER
  SEARCH DN
    URL 'ldap://voyager:389/dc=MyCompany,dc=com??sub?cn=*'
    ACCESS ACCOUNT 'cn=aseadmin, cn=Users, dc=mycompany, dc=com'
    IDENTIFIED BY 'Secret99Password'
    AUTHENTICATION URL 'ldap://voyager:389/
    CONNECTION TIMEOUT 3000
    CHECK myusername 'cn=myusername,cn=Users,dc=mycompany,dc=com';
```

**VALIDATE statement**

Validates the current database, or a single table, materialized view, or index in the current database.

**Syntax 1 - Validating a database**

```sql
VALIDATE { CHECKSUM | DATABASE }
```

**Syntax 2 - Validating tables and materialized views**

```sql
VALIDATE {
  TABLE [ owner.table-name ]
```
MATERIALIZED VIEW [owner.]materialized-view-name

[ WITH EXPRESS CHECK ]

Syntax 3 - Validating indexes

VALIDATE {
INDEX index-name
| [ INDEX ] FOREIGN KEY role-name
| [ INDEX ] PRIMARY KEY }
ON [ owner.]object-name

object-name :
table-name
| materialized-view-name

Syntax 4 - Validating text indexes

VALIDATE TEXT INDEX index-name
ON [ owner.]table-name

Remarks

Use the VALIDATE CHECKSUM statement to validate the checksum on each page of a database. The
VALIDATE CHECKSUM statement ensures that database pages have not been modified on disk. When a
database is created with checksums enabled, a checksum is calculated for each database page before it is
written to disk. VALIDATE CHECKSUM reads each database page directly from disk—not via the
database server's cache—and calculates the checksum for each page. If the calculated checksum for a
page does not match the stored checksum for that page, an error occurs and information about the invalid
page appears in the database server messages window. The VALIDATE CHECKSUM statement is not
recommended for databases that have checksums disabled because it reads the entire database from disk.

Use the VALIDATE DATABASE statement to ensure that the free map correctly identifies pages as
either allocated or free and that no BLOBs have been orphaned. VALIDATE DATABASE also performs
checksum validation and verifies that each database page belongs to the correct object. For example, on a
table page, the table ID must identify a valid table whose definition must include the current page in its set
of table pages. The VALIDATE DATABASE statement brings pages into the database server's cache in
sequential order. This results in their validation, as the database server always verifies the contents and
checksums of pages brought into the cache. If you start database validation while the database cleaner is
running, the validation does not run until the database cleaner is finished running.

The VALIDATE TABLE statement validates the specified table and all of its indexes by checking that the
set of all rows and values in the base table matches the set of rows and values contained in each index.
VALIDATE TABLE also traverses all the table's BLOBs, verifies BLOB allocation maps, and detects
orphaned BLOBs. The VALIDATE TABLE statement checks the physical structure of the table's index
pages and verifies the order of the index hash values, and the index's uniqueness requirements (if any are
specified). For foreign key indexes, unless the WITH EXPRESS CHECK clause is specified, each value
is looked up in the primary key table to verify that referential integrity is intact. Because the VALIDATE
TABLE statement, like VALIDATE DATABASE, uses the database server's cache, the database server
also verifies the checksums and basic validity of all pages in use by a table and its indexes.

The VALIDATE INDEX statement performs the same operations as the VALIDATE TABLE statement
except that it only validates the specified index and its underlying table; other indexes are not checked.
For foreign key indexes, unless the WITH EXPRESS CHECK clause is specified, each value is looked up
in the primary key table to verify that referential integrity is intact. Specifying the WITH EXPRESS CHECK clause disables referential integrity checking and can therefore significantly improve performance. If the specified index is not a foreign key index, WITH EXPRESS CHECK has no effect.

Use the VALIDATE TEXT INDEX statement to verify that the positional information for the terms in the index is intact. If the positional information is not intact, an error is generated and you must rebuild the text index. If the text index is either auto or manual, you can rebuild the text index by executing the REFRESH TEXT INDEX statement.

If the generated error concerns an immediate text index, you must drop the immediate index and create a new one.

Caution
Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, errors may be reported indicating some form of database corruption even though no corruption actually exists.

Privileges
You must have the VALIDATE ANY OBJECT system privilege.

Side effects
None.

See also
- “REFRESH TEXT INDEX statement” on page 928
- “DROP TEXT INDEX statement” on page 782
- “CREATE TEXT INDEX statement” on page 711
- “Validation utility (dbvalid)” [SQL Anywhere Server - Database Administration]
- “sa_validate system procedure” on page 1266
- “sa_clean_database system procedure” on page 1097
- “Database validation” [SQL Anywhere Server - Database Administration]
- “CREATE DATABASE statement” on page 549
- “CREATE INDEX statement” on page 599

Standards and compatibility
- SQL/2008 Vendor extension.

Example
The following example validates the Products table:

\`\`
VALIDATE TABLE GROUPO.Products;
\`\`

**WAITFOR statement**

Delays processing for the current connection for a specified amount of time or until a given time.
Syntax

\[
\text{WAITFOR } \begin{cases}
\text{DELAY } \text{time} \\
\text{TIME } \text{time} \\
\text{CHECK EVERY } \text{integer} \\
\text{AFTER MESSAGE BREAK}
\end{cases}
\]

\( \text{time : string} \)

Parameters

**DELAY clause**  If DELAY is used, processing is suspended for the given interval.

**TIME clause**  If TIME is specified, processing is suspended until the database server time reaches the time specified. If the current server time is greater than the time specified, processing is suspended until that time on the following day.

**CHECK EVERY clause**  This optional clause controls how often the WAITFOR statement wakes up. By default, it wakes up every 5 seconds. The value is in milliseconds, and the minimum value is 250 milliseconds.

**AFTER MESSAGE BREAK clause**  The WAITFOR statement can be used to wait for a message from another connection. When a message is received it is usually forwarded to the application that executed the WAITFOR statement and the WAITFOR statement continues to wait. If the AFTER MESSAGE BREAK clause is specified, when a message is received from another connection, the WAITFOR statement completes. The message text is not forwarded to the application, but it can be accessed by obtaining the value of the MessageReceived connection property.

Remarks

The WAITFOR statement wakes up periodically (every 5 seconds by default) to check if it has been canceled or if messages have been received. If neither of these has happened, the statement continues to wait.

Because scheduled events execute on their own connection, scheduled events are often a better choice than using WAITFOR TIME.

Privileges

None

Side effects

The implementation of the WAITFOR statement causes the worker servicing the statement to block while it is waiting. This reduces the number of available workers in the worker pool.

See also

- “SQL Anywhere threading” [SQL Anywhere Server - Database Administration]
- “CREATE EVENT statement” on page 570
- “MESSAGE statement” on page 899
- “MessageReceived connection property” [SQL Anywhere Server - Database Administration]
Standards and compatibility

- SQL/2008  Vendor extension.

Examples

The following example waits for three seconds:

```sql
WAITFOR DELAY '00:00:03';
```

The following example waits for 0.5 seconds (500 milliseconds):

```sql
WAITFOR DELAY '00:00:00:500';
```

The following example waits until 8 PM:

```sql
WAITFOR TIME '20:00';
```

In the following example, connection 1’s WAITFOR statement completes when it receives the message from connection 2:

```sql
// connection 1:
BEGIN
  DECLARE msg LONG VARCHAR;
  LOOP  // forever
    WAITFOR DELAY '00:05:00' AFTER MESSAGE BREAK;
    SET msg = CONNECTION_PROPERTY('MessageReceived');
    IF msg != '' THEN
      MESSAGE 'Msg: ' || msg TO CONSOLE;
    END IF;
  END LOOP
END;
// connection 2:
MESSAGE 'here it is' FOR connection 1
```

WHENEVER statement [ESQL]

Specifies error handling in embedded SQL programs.

Syntax

```
WHENEVER { SQLERROR | SQLWARNING | NOTFOUND } GOTO label

label : identifier
```

Remarks

The WHENEVER statement is used to trap errors, warnings and exceptional conditions encountered by the database when processing SQL statements. The statement can be put anywhere in an embedded SQL
program and does not generate any code. The preprocessor will generate code following each successive SQL statement. The error action remains in effect for all embedded SQL statements from the source line of the WHENEVER statement until the next WHENEVER statement with the same error condition, or the end of the source file.

Note
The error conditions are in effect based on positioning in the C language source file, not based on when the statements are executed.

The default action is CONTINUE.

This statement is provided for convenience in simple programs. Most of the time, checking the sqlcode field of the SQLCA (SQLCODE) directly is the easiest way to check error conditions. In this case, the WHENEVER statement would not be used. In fact, all the WHENEVER statement does is cause the preprocessor to generate an if (SQLCODE) test after each statement.

Privileges
None.

Side effects
None.

Standards and compatibility
- SQL/2008 An exception condition declaration made with the WHENEVER statement is a core feature of the SQL/2008 standard. The standard uses the keyword SQLEXCEPTION rather than SQLERROR. The ability to directly include C code in the WHENEVER statement, rather than merely a statement label, is a vendor extension. The action STOP is also a vendor extension.

Example
EXEC SQL WHENEVER NOTFOUND GOTO done;
EXEC SQL WHENEVER SQLERROR
  { PrintError( &sqlca );
    return( FALSE );
  };

WHILE statement [T-SQL]
Provides repeated execution of a statement or compound statement.

Syntax
WHILE search-condition statement

Remarks
The WHILE conditional affects the execution of only a single SQL statement, unless statements are grouped into a compound statement between the keywords BEGIN and END.
The BREAK statement and CONTINUE statement can be used to control execution of the statements in the compound statement. The BREAK statement terminates the loop, and execution resumes after the END keyword marking the end of the loop. The CONTINUE statement causes the WHILE loop to restart, skipping any statements after the CONTINUE.

Privileges
None.

Side effects
None.

See also
- “LOOP statement” on page 891
- “CONTINUE statement” on page 546

Standards and compatibility
- SQL/2008 Transact-SQL extension. The WHILE statement is part of optional SQL/2008 language feature P002, “Computational completeness”. The Transact-SQL variant of the WHILE statement does not include END WHILE.

Example

The following code illustrates the use of WHILE:

    WHILE ( SELECT AVG(UnitPrice) FROM GROUPO.Products ) < $30
    BEGIN
        UPDATE GROUPO.Products
        SET UnitPrice = UnitPrice + 2
        IF ( SELECT MAX(UnitPrice) FROM GROUPO.Products ) > $50
            BREAK
    END

The BREAK statement breaks the WHILE loop if the most expensive product has a price above $50. Otherwise, the loop continues until the average price is greater than or equal to $30.

WINDOW clause

Defines all or part of a window for use with window functions such as AVG and RANK in a SELECT statement.

Syntax

    WINDOW window-expression, ...

    window-expression : new-window-name AS ( window-spec )

    window-spec :
     [ existing-window-name ]
     [ PARTITION BY expression, ... ]
     [ ORDER BY expression [ ASC | DESC ], ... ]
     [ { ROWS | RANGE } { window-frame-start | window-frame-between } ]
window-frame-start :
{ UNBOUNDED PRECEDING
  | unsigned-integer PRECEDING
  | CURRENT ROW }

window-frame-between :
BETWEEN window-frame-bound1 AND window-frame-bound2

window-frame-bound :
window-frame-start
| UNBOUNDED FOLLOWING
| unsigned-integer FOLLOWING

Parameters

PARTITION BY clause The PARTITION BY clause organizes the result set into logical groups based on the unique values of the specified expression. When this clause is used with window functions, the functions are applied to each partition independently. For example, if you follow PARTITION BY with a column name, the result set is partitioned by distinct values in the column.

If this clause is omitted, the entire result set is considered a partition.

The PARTITION BY expression cannot be an integer literal.

ORDER BY clause The ORDER BY clause defines how to sort the rows in each partition of the result set. You can further control the order by specifying ASC for ascending order (the default) or DESC for descending order.

The ORDER BY expression cannot be an integer literal.

If this clause is omitted, SQL Anywhere returns rows in whatever order is most efficient, and the appearance of result sets may vary depending on when you last accessed the row.

ROWS clause and RANGE clause Use either a ROWS or RANGE clause to express the size of the window. The window size can be one, many, or all rows of a partition. You can express the size of the window as a range of data values offset from the value in the current row (RANGE), or the number of physical rows offset from the current row (ROWS).

When using the RANGE clause, you must also specify an ORDER BY clause because range calculations require values to be sorted. The ORDER BY clause for ranges must contain one expression, and that expression must result in either a date or a numeric value.

If you do not specify a ROWS or RANGE clause, the database server uses default window sizes based on whether an ORDER BY clause is present.

○ PRECEDING clause Use the PRECEDING clause to define the first row of the window using the current row as a reference point. The starting row is expressed as the number of rows preceding the current row. For example, 5 PRECEDING sets the window to start with the fifth row preceding the current row.

Use UNBOUNDED PRECEDING to set the first row in the window to be the first row in the partition.
BETWEEN clause  Use the BETWEEN clause to define the first and last row of the window, using the current row as a reference point. First and last rows are expressed as the number of rows preceding and following the current row, respectively. For example, BETWEEN 3 PRECEDING AND 5 FOLLOWING sets the window to start with the third row preceding the current row, and end with the fifth row following the current row.

Use BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING to set the first and last rows in the window to be the first and last row in the partition, respectively. This is equivalent to the default behavior if no ROW or RANGE clause is specified.

FOLLOWING clause  Use the FOLLOWING clause to define the last row of the window using the current row as a reference point. The last row is expressed as the number of rows following the current row.

Use UNBOUNDED FOLLOWING to set the last row in the window to be the last row in the partition.

Remarks
The WINDOW clause must appear before the ORDER BY clause in a SELECT statement.

With the exception of the LIST function, all aggregate functions can be used as window functions. However, ranking aggregate functions (RANK, DENSE_RANK, PERCENT_RANK, CUME_DIST, and ROW_NUMBER) require an ORDER BY clause, and do not allow a ROW or RANGE clause in the WINDOW clause or inline definition. For all other window functions, you can use any of the clauses.

Privileges
None.

See also
- “SELECT statement” on page 955
- “OLAP support” [SQL Anywhere Server - SQL Usage]
- “Window definitions” [SQL Anywhere Server - SQL Usage]
- “Window definition: Inlining using the OVER clause and WINDOW clause” [SQL Anywhere Server - SQL Usage]

Standards and compatibility
- SQL/2008  The WINDOW clause and window aggregate functions comprise SQL/2008 optional language features T611, "Elementary OLAP operations", and T612, "Advanced OLAP operations". The window functions FIRST_VALUE and LAST_VALUE are vendor extensions.

Example
The following example returns an employee's salary and the average salary for all employees in the selected state. The results are ordered by state and then by surname.

```sql
SELECT EmployeeID, Surname, Salary, State,
     AVG( Salary ) OVER Salary_Window
FROM GROUPO.Employees
WINDOW Salary_Window AS ( PARTITION BY State )
ORDER BY State, Surname;
```
WRITETEXT statement [T-SQL]

Permits non-logged updating of a CHAR, NCHAR, or BINARY column. This feature is provided solely for compatibility with Transact-SQL and its use is not recommended.

Syntax

```
WRITETEXT table-name.column-name
   text-pointer [ WITH LOG ] data
```

Remarks

Updates an existing column value. The update is not recorded in the transaction log, unless the WITH LOG option is supplied. You cannot perform WRITETEXT operations on views.

Privileges

None.

Side effects

WRITETEXT does not fire triggers, and by default WRITETEXT operations are not recorded in the transaction log.

See also

- “READTEXT statement [T-SQL]” on page 924
- “TEXTPTR function [Text and image]” on page 385

Standards and compatibility

- SQL/2008 Transact-SQL extension.

Example

The following embedded SQL code fragment illustrates the use of the WRITETEXT statement. The SELECT statement in this example returns a single row. The example replaces the contents of the Description column on the specified row with a new value.

```
EXEC SQL create variable txtptr binary(16);
EXEC SQL set txtptr =
   ( SELECT txtptr(Description)
     FROM MarketingInformation
     WHERE ProductID = '500' );
EXEC SQL writetext MarketingInformation.Description txtptr 'newdata';
```
Tables

System tables

The structure of every database is described in several system tables. System tables are owned by the user SYS. The contents of these tables can be changed only by the database server. The UPDATE, DELETE, and INSERT statements cannot be used to modify the contents of these tables. Further, the structure of these tables cannot be changed using the ALTER TABLE and DROP statements. System views are updated when a checkpoint occurs.

Data in system tables are exposed via their corresponding system views. Each system table name starts with capital i (I). Each corresponding system view has the same name but without the i at the beginning.

See also

- “System views” on page 1347

DUMMY system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Column constraint</th>
<th>Table constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>dummy_col</td>
<td>INTEGER</td>
<td>NOT NULL</td>
<td></td>
</tr>
</tbody>
</table>

The DUMMY table is provided as a read-only table that always has exactly one row. This can be useful for extracting information from the database, as in the following example that gets the current user ID and the current date from the database.

```
SELECT USER, today(*) FROM SYS.DUMMY;
SELECT USER, today(*)
```

**dummy_col**  This column is not used. It is present because a table cannot be created with no columns.

The cost of reading from the SYS.DUMMY table is less than the cost of reading from a similar user-created table because there is no latch placed on the table page of SYS.DUMMY.

Access plans are not constructed with scans of the SYS.DUMMY table. Instead, references to SYS.DUMMY are replaced with a Row Constructor algorithm, which virtualizes the table reference. This eliminates contention associated with the use of SYS.DUMMY. DUMMY still appears as the table and/or correlation name in short, long, and graphical plans.

ISYSARTICLE system table

Each row in the ISYSARTICLE system table describes an article in a publication.
See also

- “SYSARTICLE system view” on page 1348

ISYSARTICLECOL system table

Each row in the ISYSARTICLECOL system table identifies a column in an article.

See also

- “SYSARTICLECOL system view” on page 1348

ISYSATTRIBUTE system table

This table is for internal use only.

ISYSATTRIBUTENAME system table

This table is for internal use only.

ISYSCAPABILITY system table

Each row in the ISYSCAPABILITY system table identifies a capability of a remote server.

See also

- “SYSCAPABILITY system view” on page 1349

ISYSCERTIFICATE system table

Each row in the ISYSCERTIFICATE system table stores a certificate in text PEM-format.

See also

- “SYSCERTIFICATE system view” on page 1350

ISYSCHECK system table

Each row in the ISYSCHECK system table identifies a named check constraint in a table.

See also

- “SYSCHECK system view” on page 1350
ISYSCOLPERM system table
Each row in the ISYSCOLPERM system table describes an UPDATE, SELECT, or REFERENCES privilege on a column.

See also
● “SYSCOLPERM system view” on page 1351

ISYSCOLSTAT system table
The ISYSCOLSTAT system table contains the column statistics used by the optimizer.

Note
For databases created using SQL Anywhere 16 or later, this table is always encrypted to protect the data from unauthorized access.

See also
● “SYSCOLSTAT system view” on page 1351

ISYSCONSTRAINT system table
Each row in the ISYSCONSTRAINT system table describes a named constraint for all tables except the system tables.

See also
● “SYSCONSTRAINT system view” on page 1352

ISYSDEPENDENCY system table
Each row in the ISYSDEPENDENCY system table describes a table or view dependency.

See also
● “SYSDEPENDENCY system view” on page 1355

ISYSDBFILE system table
Each row in the ISYSDBFILE system table describes a dbspace.

See also
● “SYSDBFILE system view” on page 1353
**ISYSDBSPACE system table**
Each row in the ISYSDBSPACE system table describes a dbspace.

See also
- “ISYSDBSPACE system view” on page 1354

**ISYSDBSPACEPERM system table**
Each row in the ISYSDBSPACEPERM system table describes a privilege on a dbspace.

See also
- “ISYSDBSPACEPERM system view” on page 1354

**ISYSDOMAIN system table**
Each of the predefined data types (also called **domains**) is assigned a unique number. The ISYSDOMAIN table is provided for informational purposes, to show the association between these numbers and the appropriate data types. This table is never changed.

See also
- “ISYSDOMAIN system view” on page 1356

**ISYSEVENT system table**
Each row in the ISYSEVENT system table describes an event created with CREATE EVENT.

See also
- “SYSEVENT system view” on page 1356

**ISYSEXTERNENV system table**
Each row in the ISYSEXTERNENV system table describes the information needed to identify and launch each of the external environments.

See also
- “SYSEXTERNENV system view” on page 1358

**ISYSEXTERNENVOBJECT system table**
Each row in the ISYSEXTERNENVOBJECT system table describes an installed external object.
ISYSEXTERNLOGIN system table
Each row in the ISYSEXTERNLOGIN system table describes an external login for remote data access.

Note
For databases created using SQL Anywhere 16 or later, this table is always encrypted to protect the data from unauthorized access.

See also
● “SYSEXTERNENVOBJECT system view” on page 1360

ISYSFKEY system table
Each row in the ISYSFKEY system table describes a foreign key in the database

See also
● “SYSFKEY system view” on page 1361

ISYSHISTORY system table
Each row in the ISYSHISTORY system table indicates a time in which the database was started with a different version of the software and/or on a different platform.

See also
● “SYSHISTORY system view” on page 1362

ISYSIDX system table
Each row in the ISYSIDX system table describes an index in the database.

See also
● “SYSIDX system view” on page 1364

ISYSIDXCOL system table
Each row in the ISYSIDXCOL system table describes a column in an index.
ISYSJAR system table
Each row in the ISYSJAR system table defines a JAR file in the system.

See also
● “SYSJAR system view” on page 1367

ISYSJARCOMPONENT system table
Each row in the ISYSJAR system table defines a JAR file component.

See also
● “SYSJARCOMPONENT system view” on page 1367

ISYSJAVACLASS system table
Each row in the ISYSJAVACLASS system table describes a Java class.

See also
● “SYSJAVACLASS system view” on page 1368

ISYSLDAPSERVER system table
The SYSLDAPSERVER system view contains one row for each LDAP SERVER object configured in the database. An LDAP SERVER object contains the configuration information necessary to connect to an LDAP server outside of SQL Anywhere.

See also
● “SYSLDAPSERVER system view” on page 1369

ISYSLOGINMAP system table
The ISYSLOGINMAP system table contains all the User Profile names that can be used to connect to the database using either an integrated login or a Kerberos login.

See also
● “SYSLOGINMAP system view” on page 1370
ISYSLOGINPOLICY system table
Each row in the ISYSLOGINPOLICY system table describes a login policy.

See also
- “SYSLOGINPOLICY system view” on page 1371

ISYSLOGINPOLICYOPTION system table
Each row in the ISYSLOGINPOLICYOPTION system table describes an option for a login policy.

See also
- “SYSLOGINPOLICYOPTION system view” on page 1371

ISYSMIRROROPTION system table
Each row in the ISYSMIRROROPTION system table describes an option for a mirror server.

See also
- “SYSMIRROROPTION system view” on page 1372

ISYSMIRRORSERVER system table
Each row in the ISYSMIRRORSERVER system table describes an option for a mirror server.

See also
- “SYSMIRRORSERVER system view” on page 1372

ISYSMIRRORSERVEROPTION system table
Each row in the ISYSMIRRORSERVEROPTION system table describes an option for a mirror server.

See also
- “SYSMIRRORSERVEROPTION system view” on page 1372

ISYSMVOPTION system table
Each row in the ISYSMVOPTION system table gives the value of a creation option for a materialized view or text index in the database.

See also
- “SYSMVOPTION system view” on page 1373
**ISYSMVOPTIONNAME system table**

Each row in the ISYSMVOPTIONNAME system table provides the name of a creation option listed in ISYSMVOPTION for a materialized view or text index.

See also
- “SYSMVOPTIONNAME system view” on page 1374

**ISYSOBJECT system table**

Each row in the ISYSOBJECT system view describes an object in the database. Examples of database objects include tables, views, columns, indexes, and procedures.

See also
- “SYSOBJECT system view” on page 1375

**ISYSOPTION system table**

Each row in the ISYSOPTION system table describes the settings for an option for one user ID. Options settings are stored in the ISYSOPTION table by the SET statement, and each user can have their own setting for each option.

See also
- “SYSOPTION system view” on page 1375

**ISYSOPTSTAT system table**

The ISYSOPTSTAT system table stores the cost model calibration information as computed by the ALTER DATABASE CALIBRATE statement.

See also
- “SYSOPTSTAT system view” on page 1376

**ISYSPHYSIDX system table**

Each row in the ISYSPHYSIDX system table describes a physical index in the database.

See also
- “SYSPHYSIDX system view” on page 1376
**ISYSPROCEDURE system table**

Each row in the ISYSPROCEDURE system table describes a procedure in the database.

See also
- “ISYSPROCEDURE system view” on page 1377

**ISYSPROCPARM system table**

Each row in the ISYSPROCPARM system table describes a parameter to a procedure in the database.

See also
- “ISYSPROCPARM system view” on page 1378

**ISYSROCPERM system table**

Each row in the ISYSROCPERM system table describes a user granted privilege to call one procedure.

See also
- “ISYSROCPERM system view” on page 1380

**ISYSPROXYTAB system table**

Each row in the ISYSPROXYTAB system table describes a proxy table.

See also
- “ISYSPROXYTAB system view” on page 1380

**ISYSPUBLICATION system table**

Each row in the ISYSPUBLICATION system table describes a SQL Remote or MobiLink publication.

See also
- “ISYSPUBLICATION system view” on page 1381

**ISYSREMARK system table**

Each row in the ISYSREMARK system table describes a remark (or comment) for an object.

See also
- “ISYSREMARK system view” on page 1382
ISYSREMOOTEOPITION system table

Each row in the ISYSREMOOTEOPITION system table describes the values of a SQL Remote message link parameter.

See also

- “SYSREMOOTEOPITION system view” on page 1383

ISYSREMOOTEOPITIONTYPE system table

Each row in the ISYSREMOOTEOPITIONTYPE system table describes one of the SQL Remote message link parameters.

See also

- “SYSREMOOTEOPITIONTYPE system view” on page 1383

ISYSREMOETETYPE system table

The ISYSREMOETETYPE system table contains information about SQL Remote.

See also

- “SYSREMOETETYPE system view” on page 1383

ISYSREMOTEUSER system table

Each row in the ISYSREMOTEUSER system table describes a user ID with REMOTE privileges (a subscriber), together with the status of SQL Remote messages that were sent to and from that user.

See also

- “SYSREMOTEUSER system view” on page 1384
- “REMOTE RESET statement [SQL Remote]” on page 932

ISYSROLEGRANT system table

Rows in the ISYSROLEGRANT system table provide information about role membership and type of membership.

See also

- “SYSROLEGRANT system view” on page 1386
- “SYSROLEGRANTS consolidated view” on page 1387
ISYSROLEGRANTEXT system table

The ISYSROLEGRANTEXT system table stores the syntax extensions for SET USER and CHANGE PASSWORD system privilege.

See also
● “SYSROLEGRANTEXT system view” on page 137

ISYSSCHEDULE system table

Each row in the ISYSSCHEDULE system table describes a time at which an event is to fire, as specified by the SCHEDULE clause of CREATE EVENT.

See also
● “SYSSCHEDULE system view” on page 138

ISYSSEQUENCE system table

The ISYSSEQUENCE system table contains one row for each user-defined sequence.

See also
● “SYSSEQUENCE system view” on page 139

ISYSSEQUENCEPERM system table

The ISYSSEQUENCEPERM system table records the privileges that users or groups hold on sequences.

See also
● “SYSSEQUENCEPERM system view” on page 139

ISYSSERVER system table

Each row in the ISYSSERVER system table describes a remote server.

See also
● “SYSSERVER system view” on page 139

ISYSSOURCE system table

Each row in the ISYSSOURCE system view contains the source for an object listed in the ISYSOBJECT system table.
ISYSSPATIALREFERENCESYSTEM system table

Each row in the ISYSSPATIALREFERENCESYSTEM system table describes a spatial reference system defined in the database.

See also
● “SYSSPATIALREFERENCESYSTEM system view” on page 1392

ISYSQLSERVERTYPE system table

The ISYSQLSERVERTYPE system table contains information relating to compatibility with Adaptive Server Enterprise.

See also
● “SYSQLSERVERTYPE system view” on page 1395

ISYSSUBSCRIPTION system table

Each row in the ISYSSUBSCRIPTION system table describes a subscription from one user ID (which must have REMOTE privileges) to one publication.

See also
● “SYSSUBSCRIPTION system view” on page 1395

ISYSSYNC system table

This table contains information relating to MobiLink synchronization.

See also
● “SYSSYNC system view” on page 1396

ISYSSYNCPROFILE system table

This table contains information relating to synchronization profiles for MobiLink.

See also
● “SYSSYNCPROFILE system view” on page 1397
**ISYSSYNCScrip**t system table

This table contains information relating to MobiLink synchronization scripts.

See also

- “SYSSYNCScrip**t system view” on page 1398

**ISYSTAB** system table

Each row in the ISYSTAB system table describes one table in the database.

See also

- “SYTAB system view” on page 1399

**ISYSTABCol** system table

Each row in the ISYSTABCol system table describes a column of a table in the database.

See also

- “SYTABCol system view” on page 1401

**ISYSTEXTCONFIG** system table

Each row in the ISYSTEXTCONFIG system table describes a text configuration, for use with the full text search feature.

See also

- “SYSTEXTCONFIG system view” on page 1405

**ISYSTEXTIDX** system table

Each row in the ISYSTEXTIDX system table describes a text index, for use with the full text search feature.

See also

- “SYSTEXTIDX system view” on page 1407

**ISYSTEXTIDXTAB** system table

Each row in the ISYSTEXTIDXTAB system table describes a text index, for use with the full text search feature.
ISYSTABLEPERM system table
Each row in the ISYSTABLEPERM system table corresponds to one table, one user ID granting the privilege (grantor) and one user ID granted the privilege (grantee).

See also
● “SYSTABLEPERM system view” on page 1404

ISYSTRIGGER system table
Each row in the ISYSTRIGGER system table describes a trigger in the database.

See also
● “SYSTRIGGER system view” on page 1409

ISYSTYPEMAP system table
The ISYSTYPEMAP system table contains the compatibility mapping values for the ISYSSQLSERVERTYPE system table.

See also
● “SYSTYPEMAP system view” on page 1411

ISYSUNITOFMEASURE system table
Each row in the ISYSUNITOFMEASURE system table describes a unit of measure defined in the database.

See also
● “SYSUNITOFMEASURE system view” on page 1411

ISYSUSER system table
Each row in the ISYSUSER system table describes a user in the system.

Note
As of SQL Anywhere 12, this table is always encrypted to protect the data from unauthorized access.
Diagnostic tracing tables

See also
- “SYSUSER system view” on page 1412

ISYSUSERMESSAGE system table
Each row in the ISYSUSERMESSAGE system table holds a user-defined message for an error condition.

See also
- “ISYSUSERMESSAGE system view” on page 1414

ISYSUSERTYPE system table
Each row in the ISYSUSERTYPE system table describes a user-defined data type.

See also
- “ISYSUSERTYPE system view” on page 1415

ISYSVIEW system table
Each row in the ISYSVIEW system table describes a view in the database.

See also
- “ISYSVIEW system view” on page 1416

ISYSWEBSERVICE system table
Each row in the ISYSWEBSERVICE system table describes a web service.

See also
- “ISYSWEBSERVICE system view” on page 1418

Diagnostic tracing tables
Following are the main tables that are used for application profiling and diagnostic tracing. These tables are owned by the dbo user. For many of these tables, there exists a global shared temporary table with a similar name and schema. For example, the sa_diagnostic_blocking table has a global temporary table counterpart, sa_tmp_diagnostic_blocking table, which has the same schema. During a tracing session, diagnostic data is written to these temporary tables. Because temporary tables are not logged, they provide superior performance during a tracing session, where it is important to minimize the impact on the server.
See also

- “Application profiling” [SQL Anywhere Server - SQL Usage]
- “Diagnostic tracing” [SQL Anywhere Server - SQL Usage]

**sa_diagnostic_auxiliary_catalog table**

The sa_diagnostic_auxiliary_catalog table is owned by the dbo user, and is used to map database objects between the production database and tracing database. Objects include user tables, procedures, and functions. This table is used primarily by the Index Consultant and the TRACED_PLAN function.

**Columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>original_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of this object in the main tracing database.</td>
</tr>
<tr>
<td>local_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of this object in the auxiliary tracing database.</td>
</tr>
<tr>
<td>pages_if_table</td>
<td>UNSIGNED INT</td>
<td>If the object is a table, this is the number of pages in the table. If the object is not a table, this value is NULL.</td>
</tr>
<tr>
<td>rows_if_table</td>
<td>UNSIGNED BIGINT</td>
<td>If the object is a table, this is the number of rows in the table. If the object is not a table, this value is NULL.</td>
</tr>
</tbody>
</table>

See also

- “TRACED_PLAN function [Miscellaneous]” on page 390
- “Index Consultant” [SQL Anywhere Server - SQL Usage]

**sa_diagnostic_blocking table**

The sa_diagnostic_blocking table is owned by the dbo user, and records blocking events. If logging of blocking events is enabled, a row is inserted in this table each time a connection is blocked while trying to access a resource. Typically, this is caused by either a table or a row lock. A large number of blocks may indicate that you should examine the concurrency in your application to reduce contention for tables and rows.

There are two versions of this table: sa_diagnostic_blocking, and sa_tmp_diagnostic_blocking.

**Columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED</td>
<td>A number uniquely identifying the logging session during which the diagnostic information was gathered.</td>
</tr>
<tr>
<td>INT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Column name</td>
<td>Column type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lock_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the lock that caused the blocking if a row or table lock caused the block, otherwise NULL.</td>
</tr>
<tr>
<td>request_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the request that was blocked if the block did not occur because of a cursor, otherwise NULL. This value corresponds to the ID assigned to the request in sa_diagnostic_request.</td>
</tr>
<tr>
<td>cursor_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the cursor if the block occurred because of a cursor, otherwise NULL. This value corresponds to the ID assigned to the cursor in sa_diagnostic_cursor.</td>
</tr>
<tr>
<td>original_table_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>If the block occurred because of a table lock, the ID of the table on which the block occurred, otherwise NULL.</td>
</tr>
<tr>
<td>rowid</td>
<td>UNSIGNED BIGINT</td>
<td>If the block occurred because of a row lock, the ID of the row on which the block occurred, otherwise NULL.</td>
</tr>
<tr>
<td>block_time</td>
<td>TIMESTAMP</td>
<td>The time at which the block occurred.</td>
</tr>
<tr>
<td>unblock_time</td>
<td>TIMESTAMP</td>
<td>The time at which the block ended.</td>
</tr>
<tr>
<td>blocked_by</td>
<td>UNSIGNED INT</td>
<td>The ID of the connection that held the lock, causing the block.</td>
</tr>
</tbody>
</table>

See also
- “Transaction blocking and deadlock” [SQL Anywhere Server - SQL Usage]
- “How locking works” [SQL Anywhere Server - SQL Usage]

**sa_diagnostic_cachecontents table**

The sa_diagnostic_cachecontents table is owned by the dbo user. When diagnostic tracing is enabled, periodic snapshots of the cache contents are taken. The sa_diagnostic_cachecontents table records the number of table pages for each table in the cache at the time the snapshot was taken, and the number of rows in each table. The optimizer can use this information to recreate the conditions under which a query was originally optimized, and then make optimization decisions.

Data in the sa_diagnostic_cachecontents table is updated every 20 seconds, as long as there is query activity.

There are two versions of this table: sa_diagnostic_cachecontents, and sa_tmp_diagnostic_cachecontents.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>A number uniquely identifying the logging session during which the diagnostic information was gathered.</td>
</tr>
<tr>
<td>&quot;time&quot;</td>
<td>TIMESTAMP</td>
<td>The time at which the snapshot of the cache was taken.</td>
</tr>
<tr>
<td>original_table_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of each table represented in the snapshot.</td>
</tr>
<tr>
<td>pages_in_cache</td>
<td>UNSIGNED INT</td>
<td>For a specified table in the snapshot, the total number of pages in cache at the moment of the snapshot.</td>
</tr>
<tr>
<td>num_table_pages</td>
<td>UNSIGNED INT</td>
<td>For a specified table in the snapshot, the total number of pages for the table.</td>
</tr>
<tr>
<td>num_table_rows</td>
<td>UNSIGNED BIGINT</td>
<td>For a specified table in the snapshot, the total number of rows in the table.</td>
</tr>
</tbody>
</table>

**sa_diagnostic_connection table**

The sa_diagnostic_connection table is owned by the dbo user, and has one row for every database connection that is active during the logging session. Connect and disconnect times, if they occur within the logging session, can be derived from the sa_diagnostic_request table.

Most of the values in this table mirror values of connection properties.

There are two versions of this table: sa_diagnostic_connection, and sa_tmp_diagnostic_connection.

### Columns
<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>A number uniquely identifying the logging session during which the diagnostic information was gathered.</td>
</tr>
<tr>
<td>connection_number</td>
<td>UNSIGNED INT</td>
<td>A number assigned by the database server to identify the connection of a specific user to the database. This value reflects the value of the Number connection property.</td>
</tr>
<tr>
<td>connection_name</td>
<td>LONG VARCHAR</td>
<td>Optional name property for the connection. This value reflects the value of the Name connection property.</td>
</tr>
<tr>
<td>Column name</td>
<td>Column type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>user_name</td>
<td>LONG VARCHAR</td>
<td>The name of the user connected to the database.</td>
</tr>
<tr>
<td>comm_link</td>
<td>CHAR(40)</td>
<td>Specifies the client-side network protocol options. This value reflects the value of the CommLinks connection property.</td>
</tr>
<tr>
<td>node_address</td>
<td>LONG VARCHAR</td>
<td>The node for the client in a client/server connection. This value reflects the value of the NodeAddress connection property.</td>
</tr>
<tr>
<td>appinfo</td>
<td>LONG VARCHAR</td>
<td>Information about the client process, such as the IP address of the client computer, the operating system it is running on, and so on. This value reflects the value of the AppInfo connection property.</td>
</tr>
</tbody>
</table>

See also

- “List of connection properties” [SQL Anywhere Server - Database Administration]

**sa_diagnostic_cursor table**

The sa_diagnostic_cursor table is owned by the dbo user. Each row describes either an internal or external cursor opened during the logging session.

There are two versions of this table: sa_diagnostic_cursor, and sa_tmp_diagnostic_cursor.

**Columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>A number uniquely identifying the logging session during which the diagnostic information was gathered.</td>
</tr>
<tr>
<td>cursor_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique number identifying the cursor.</td>
</tr>
<tr>
<td>query_id</td>
<td>UNSIGNED BIGINT</td>
<td>Identifies the query over which this cursor ranges.</td>
</tr>
<tr>
<td>isolation_level</td>
<td>TINYINT</td>
<td>Isolation level at which this cursor was opened.</td>
</tr>
</tbody>
</table>
### sa_diagnostic_deadlock table

The sa_diagnostic_deadlock table is owned by the dbo user. When diagnostic tracing is enabled and is set to include tracing of deadlock events, a set of rows is inserted into this table every time a deadlock occurs (one row for each connection that was part of the deadlock is inserted). The set of all rows that comprise a single deadlock event is uniquely identified by a snapshot_id.

#### Columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>A number uniquely identifying the logging session during which the diagnostic information was gathered.</td>
</tr>
<tr>
<td>snapshot_id</td>
<td>UNSIGNED BIGINT</td>
<td>A number identifying which deadlock event this row is a part of. This column has nothing to do with snapshot isolation.</td>
</tr>
<tr>
<td>snapshot_at</td>
<td>TIMESTAMP</td>
<td>The time at which the deadlock occurred.</td>
</tr>
<tr>
<td>waiter</td>
<td>UNSIGNED INT</td>
<td>The connection number of the connection that this row represents.</td>
</tr>
</tbody>
</table>
### Column name | Column type | Description
---|---|---
request_id | UNSIGNED BIGINT | The ID of the request that this connection was processing when the deadlock occurred.
original_table_object_id | UNSIGNED BIGINT | The object ID of the table on which this connection was blocked.
rowid | UNSIGNED BIGINT | The record ID of the row on which this connection was blocked.
owner | UNSIGNED INT | The connection number of the connection that locked the desired row.
rollback_operation_count | UNSIGNED INT | The number of uncommitted operations.

See also
- “Transaction blocking and deadlock” [SQL Anywhere Server - SQL Usage]

### sa_diagnostic_hostvariable table
The sa_diagnostic_hostvariable table is owned by the dbo user, and contains the values of host variables used by the specified cursor.

There are two versions of this table: sa_diagnostic_hostvariable, and sa_tmp_diagnostic_hostvariable.

### Columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>A number uniquely identifying the logging session during which the diagnostic information was gathered.</td>
</tr>
<tr>
<td>request_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the request to which the host variables belong.</td>
</tr>
<tr>
<td>cursor_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the cursor to which the host variables pertain.</td>
</tr>
<tr>
<td>hostvar_num</td>
<td>UNSIGNED SMALLINT</td>
<td>The ordinal position of the host variable in the SQL statement.</td>
</tr>
<tr>
<td>hostvar_type</td>
<td>UNSIGNED TINYINT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>hostvar_value</td>
<td>LONG NVARCHAR</td>
<td>A string representing the value of the host variable. Even if the host variable is an integer or a float, the value is still represented here as a string.</td>
</tr>
</tbody>
</table>
The sa_diagnostic_internalvariable table is owned by the dbo user, and contains the values of internal (local) variables used by a given statement. This table is primarily used by the Index Consultant, and the traced_plan function.

There are two versions of this table: sa_diagnostic_internalvariable, and sa_tmp_diagnostic_internalvariable.

### Columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>A number uniquely identifying the logging session during which the diagnostic information was gathered.</td>
</tr>
<tr>
<td>request_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the request that contains the internal variable.</td>
</tr>
<tr>
<td>rowvariable_id</td>
<td>UNSIGNED INT</td>
<td>The column number in the row variable of this value.</td>
</tr>
<tr>
<td>variable_domain</td>
<td>UNSIGNED SMALL-INT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>variable_name</td>
<td>CHAR(128)</td>
<td>The name of the internal variable.</td>
</tr>
<tr>
<td>variable_value</td>
<td>LONG NVARCHAR</td>
<td>A string representing the value of the internal variable.</td>
</tr>
</tbody>
</table>

See also

- “Local variables” on page 80

The sa_diagnostic_query table is owned by the dbo user, and stores optimization information for queries, especially the context in which they were optimized. A row in this table represents an invocation of the optimizer for a query. Plans captured at optimization time are stored here.

Some of the values in this table mirror database option values.

There are two versions of this table: sa_diagnostic_query, and sa_tmp_diagnostic_query.
### Columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>The ID of the logging session during which the query or request occurred.</td>
</tr>
<tr>
<td>query_id</td>
<td>UNSIGNED BIGINT</td>
<td>A number uniquely identifying the query.</td>
</tr>
<tr>
<td>statement_id</td>
<td>UNSIGNED BIGINT</td>
<td>A number uniquely identifying a statement in a query.</td>
</tr>
<tr>
<td>user_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the user under which this query was executed. If the query was run from a procedure, this would be the user ID of the procedure owner.</td>
</tr>
<tr>
<td>start_time</td>
<td>TIME-STAMP</td>
<td>The time at which this query was optimized.</td>
</tr>
<tr>
<td>cache_size_bytes</td>
<td>UNSIGNED BIGINT</td>
<td>The size, in bytes, of the cache at the time this query was optimized.</td>
</tr>
<tr>
<td>optimization_goal</td>
<td>TINYINT</td>
<td>Determines whether query processing is optimized towards returning the first row quickly, or minimizing the cost of returning the complete result set. This value reflects the value of the optimization_goal database option.</td>
</tr>
<tr>
<td>optimization_level</td>
<td>TINYINT</td>
<td>Controls the amount of effort made by the SQL Anywhere query optimizer to find an access plan for a SQL statement. This value reflects the value of the optimization_level database option.</td>
</tr>
<tr>
<td>user_estimates</td>
<td>TINYINT</td>
<td>Controls whether user selectivity estimates in query predicates are respected or ignored by the query optimizer. This value reflects the value of the user_estimates database option.</td>
</tr>
<tr>
<td>optimization_workload</td>
<td>TINYINT</td>
<td>Determines whether query processing is optimized towards a workload that is a mix of updates and reads or a workload that is predominantly read-based. This value reflects the value of the optimization_workload database option.</td>
</tr>
<tr>
<td>available_requests</td>
<td>TINYINT</td>
<td>Used internally to compute the level of intra-query parallelism.</td>
</tr>
<tr>
<td>active_requests</td>
<td>TINYINT</td>
<td>Used internally to compute the level of intra-query parallelism.</td>
</tr>
</tbody>
</table>
### sa_diagnostic_request table

The sa_diagnostic_request table is owned by the dbo user, and is the master table for all requests. A request is an event related to query processing and generally includes:

- connect or disconnect events
- statement executions
- statement preparations
- open or drop cursor events

There are two versions of this table: sa_diagnostic_request and sa_tmp_diagnostic_request.
Columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>The logging session during which the request occurred.</td>
</tr>
<tr>
<td>request_id</td>
<td>UNSIGNED BIGINT</td>
<td>A number uniquely identifying the request.</td>
</tr>
<tr>
<td>start_time</td>
<td>TIMESTAMP</td>
<td>The time at which the event started.</td>
</tr>
<tr>
<td>finish_time</td>
<td>TIMESTAMP</td>
<td>For statement execution, the time when the statement completed; otherwise, NULL.</td>
</tr>
<tr>
<td>duration_ms</td>
<td>UNSIGNED INT</td>
<td>The duration of the event in milliseconds.</td>
</tr>
<tr>
<td>connection_number</td>
<td>UNSIGNED INT</td>
<td>The ID of the connection that caused the event to happen.</td>
</tr>
<tr>
<td>request_type</td>
<td>UNSIGNED SMALLINT</td>
<td>The type of request. Values include: New diagnostic tracing session started, SQL Statement executed, Cursor opened, Cursor closed, Client connect, Client disconnect, and Checkpoint.</td>
</tr>
<tr>
<td>statement_id</td>
<td>UNSIGNED BIGINT</td>
<td>If the event was statement-related, the ID assigned to the statement for tracing purposes.</td>
</tr>
<tr>
<td>query_id</td>
<td>UNSIGNED BIGINT</td>
<td>If the event was query-related, the ID assigned to the query for tracing purposes.</td>
</tr>
<tr>
<td>cursor_id</td>
<td>UNSIGNED BIGINT</td>
<td>If the event was cursor-related, the ID assigned to the cursor for tracing purposes.</td>
</tr>
<tr>
<td>sql_code</td>
<td>SMALLINT</td>
<td>Since rows in this table represent operations on statements, cursors, or queries, most return a SQL code. This column contains the SQL code returned. If a SQL code of 0 is returned, the column contains NULL.</td>
</tr>
</tbody>
</table>

**sa_diagnostic_statement table**

The sa_diagnostic_statement table is owned by the dbo user, and stores the text of statements. A row in this table represents a SQL statement that was executed by the server. Such statements may have been issued by an external source, such as a client request, or by an internal source such as a procedure, trigger, or user-defined function. Internal statements only appear here once per session.

There are two versions of this table: sa_diagnostic_statement, and sa_tmp_diagnostic_statement.
Columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>The logging session during which the statement was submitted.</td>
</tr>
<tr>
<td>statement_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique number assigned to the statement for tracing purposes.</td>
</tr>
<tr>
<td>database_object</td>
<td>UNSIGNED BIGINT</td>
<td>If the statement came from a procedure, trigger, or function, this is the ID as specified in the ISYSOBJECT system table.</td>
</tr>
<tr>
<td>line_number</td>
<td>UNSIGNED SMALLINT</td>
<td>If the statement formed part of a compound statement, this reflects the ordinal position of the statement within the compound statement.</td>
</tr>
<tr>
<td>signature</td>
<td>UNSIGNED INT</td>
<td>Used internally to group similar queries.</td>
</tr>
<tr>
<td>statement_text</td>
<td>LONG VARCHAR</td>
<td>The statement text.</td>
</tr>
</tbody>
</table>

**sa_diagnostic_statistics table**

The sa_diagnostic_statistics table is owned by the dbo user, and contains a history of performance counters maintained in the server. Each row represents the value of a given performance counter at a given moment in time.

There are two versions of this table: sa_diagnostic_statistics, and sa_tmp_diagnostic_statistics.

Columns

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging_session_id</td>
<td>UNSIGNED INT</td>
<td>A number uniquely identifying the logging session during which the diagnostic information was gathered.</td>
</tr>
<tr>
<td>&quot;time&quot;</td>
<td>TIMESTAMP</td>
<td>The time at which the performance counter value was captured.</td>
</tr>
<tr>
<td>counter_id</td>
<td>UNSIGNED SMALLINT</td>
<td>A number uniquely identifying the performance counter. You can get the name of the property that this counter_id represents using the PROPERTY_NAME function.</td>
</tr>
<tr>
<td>Column name</td>
<td>Column type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>type</td>
<td>TINYINT</td>
<td>Indicates whether this is a database, server, or connection statistic. Possible values are 0 for server, 1 for database, 2 for connection, and 4 for external database.</td>
</tr>
<tr>
<td>connection_number</td>
<td>UNSIGNED INT</td>
<td>For a connection statistic, the connection number from which this property was captured. For an extended database statistic, the file number for the file from which this property was captured. Otherwise, the value is 0.</td>
</tr>
<tr>
<td>counter_value</td>
<td>UNSIGNED INT</td>
<td>The value of the performance counter.</td>
</tr>
</tbody>
</table>

See also
- “PROPERTY_NAME function [System]” on page 322

**sa_diagnostic_tracing_level table**

The sa_diagnostic_tracing_level table is owned by the dbo user, and each row in this table is a condition that determines what kind of diagnostic information to send to the tracing database. If a piece of logging data meets the conditions of one or more rows in this table, then the corresponding data is logged.

Data in this table is populated using the CONNECT TRACING or REFRESH TRACING LEVEL statements.

**Columns**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>UNSIGNED INT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>Column name</td>
<td>Column type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>scope</td>
<td>CHAR(32)</td>
<td>The scope of the diagnostic tracing, as listed below.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● DATABASE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ORIGIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● CONNECTION_NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● CONNECTION_NUMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● FUNCTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● PROCEDURE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● EVENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● TRIGGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● TABLE</td>
</tr>
<tr>
<td>identifier</td>
<td>CHAR(128)</td>
<td>The identifier for the scope. This value changes, depending on the specified scope. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● if scope is DATABASE, identifier may not be present.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● if scope is ORIGIN, identifier must be either Internal or External.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● if scope is USER, identifier is the ID of the user.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● if scope is CONNECTION_NAME, or CONNECTION_NUMBER, identifier is the name or number, respectively, for the connection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● if scope is FUNCTION, PROCEDURE, EVENT, TRIGGER, or TABLE, identifier is the fully qualified identifier for the object.</td>
</tr>
<tr>
<td>trace_type</td>
<td>CHAR(32)</td>
<td>The type of data to trace for the specified scope, as listed below.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● VOLATILE_STATISTICS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● NONVOLATILE_STATISTICS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● CONNECTION_STATISTICS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● BLOCKING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● PLANS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● PLANS_WITH_STATISTICS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● STATEMENTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● STATEMENTS_WITH_VARIABLES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● OPTIMIZATION_LOGGING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● OPTIMIZATION_LOGGING_WITH_PLANS</td>
</tr>
<tr>
<td><strong>Column name</strong></td>
<td><strong>Column type</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>
| trace_condition | CHAR(32) | Applies only to plans, and controls whether to trace large, expensive queries, or queries for which the optimizer did not make optimal choices. Possible values are listed below.  
  ● NONE, or NULL  
  ● SAMPLE_EVERY  
  ● ABSOLUTE_COST  
  ● RELATIVE_COST_DIFFERENCE |
| value | UNSIGNED INT | The value associated with the condition. For example, if condition is SAMPLE_EVERY, the condition_value would be a positive integer reflecting time in milliseconds. Additional rules are as follows:  
  ● If condition is NULL or NONE, there is no condition_value.  
  ● If condition is ABSOLUTE_COST, condition_value reflects the total actual cost of executing the statement, in milliseconds.  
  ● If condition is RELATIVE_COST_DIFFERENCE, condition_value reflects the cost of executing, as a percentage of the estimated cost. |
| enabled | BIT | Whether the row is enabled. That is, whether the tracing settings in the row are active. 1 is enabled; 0 is disabled. |

**See also**
- “Diagnostic tracing conditions” [SQL Anywhere Server - SQL Usage]
- “Diagnostic tracing scopes” [SQL Anywhere Server - SQL Usage]
- “ATTACH TRACING statement” on page 514
- “REFRESH TRACING LEVEL statement” on page 930

**Other tables**

Following is information about other tables such as system tables used by Java in the database and SQL Remote.

**RowGenerator table (dbo)**

The dbo.RowGenerator table is provided as a read-only table that has 255 rows. This table can be useful for queries which produce small result sets and which need a range of numeric values.

The RowGenerator table is used by system procedures and views, and should not be modified in any way.

You can also use the sa_rowgenerator system procedure to generate a range of numeric values.
### Java system tables

The system tables that are used for Java are listed below. Foreign key relations between tables are indicated by arrows: the arrow leads from the foreign table to the primary table.

### MobiLink system tables

For information about the MobiLink system tables, see “MobiLink server system tables” [*MobiLink - Server Administration*].

### SQL Remote system tables

For information about the SQL Remote system tables, see “SQL Remote system tables” [*SQL Remote*].

### UltraLite system tables

For information about the UltraLite system tables, see “UltraLite system tables” [*UltraLite - Database Management and Reference*].
System procedures

A few system procedures, such as sa_get_table_definition, are implemented as functions. However, because they are used in the same context and manner as system procedures, they are included with the system procedures, and their naming is similar to the system procedures (sa_xxx). All system procedures can be enabled and disabled as secure features. You cannot call external functions on Windows Mobile.

Viewing details about system procedures and functions

Database administrators can access the definitions for system procedures and functions from Sybase Central.

Prerequisites

There are no prerequisites required.

View details about system procedures and functions

1. Use the SQL Anywhere 16 plug-in to connect to the database.
2. Right-click the database and then click Configure Owner Filter.
3. Click DBO and then click OK.
4. In the left pane, double-click Procedures & Functions.
5. In the left pane, select the procedure and in the right pane click the SQL tab.

Results

The procedure or function definition appears on the SQL tab.

Web services system procedures

The following system procedures are for use with web services:

- “sa_http_header_info system procedure”
- “sa_http_php_page system procedure”
- “sa_http_php_page_interpreted system procedure”
- “sa_http_variable_info system procedure”
- “sa_set_http_header system procedure”
- “sa_set_http_option system procedure”
- “sa_set_soap_header system procedure”
See also

- “Web services functions” on page 151
- “SQL Anywhere as an HTTP web server” [SQL Anywhere Server - Programming]
- “-xs database server option” [SQL Anywhere Server - Database Administration]

Roles and privileges system procedures

The following system procedures are for use with roles and privileges:

- “sp_displayroles system procedure” on page 1276
- “sp_has_role system procedure” on page 1284
- “sp_objectpermission system procedure” on page 1292
- “sp_proc_priv system procedure” on page 1297
- “sp_sys_priv_role_info system procedure” on page 1311

MAPI and SMTP system procedures

SQL Anywhere includes system procedures for sending electronic mail using the Microsoft Messaging API standard (MAPI) or the Internet standard Simple Mail Transfer Protocol (SMTP). These system procedures are implemented as extended system procedures: each procedure calls a function in an external DLL.

These procedures are owned by the dbo role. Users must be granted the EXECUTE privilege on these system procedures before they can use them.

To use the MAPI or SMTP system procedures, a MAPI or SMTP email system must be accessible from the database server computer.

The MAPI and SMTP system procedures are:

- xp_startmail Starts a mail session in a specified mail account by logging on to the MAPI message system.
- xp_startsmtp Starts a mail session in a specified mail account by logging on to the SMTP message system.
- xp_sendmail Sends a mail message to specified users.
- xp_stopmail Closes the MAPI mail session.
- xp_stopsmtp Closes the SMTP mail session.
- xp_get_mail_error_code Returns information about the most recent SMTP or MAPI error.
- xp_get_mail_error_text Returns the text of the most recent SMTP error.

Return codes for MAPI and SMTP system procedures

The following codes can be returned by either the MAPI or the SMTP system procedures:
<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Unknown error¹</td>
</tr>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>1</td>
<td>An invalid parameter was supplied</td>
</tr>
<tr>
<td>2</td>
<td>Out of memory</td>
</tr>
<tr>
<td>3</td>
<td>xp_startmail or xp_startsmtp was not called</td>
</tr>
<tr>
<td>4</td>
<td>Bad host name</td>
</tr>
<tr>
<td>5</td>
<td>Connect error¹</td>
</tr>
<tr>
<td>6</td>
<td>Secure connection error¹</td>
</tr>
<tr>
<td>7</td>
<td>MAPI functions are not available¹</td>
</tr>
</tbody>
</table>

¹ Use the xp_get_mail_error_code system procedure to get additional information about a return code.

**Example**

The following procedure notifies a set of people that a backup has been completed. It does not check return codes.

```sql
CREATE PROCEDURE notify_backup( )
BEGIN
    CALL xp_startmail( mail_user='ServerAccount', mail_password='ServerPassword' );
    CALL xp_sendmail( recipient='IS Group', subject='Backup', "message"='Backup completed' );
    CALL xp_stopmail( );
END;
```

The following procedure notifies a set of people that a backup has been completed and includes error checking.

```sql
CREATE OR REPLACE PROCEDURE notify_backup_with_error_check( )
BEGIN
    DECLARE result_code INTEGER;
    DECLARE error_code INTEGER;
    DECLARE error_text LONG VARCHAR;
    SET result_code = xp_startmail( 'ServerAccount', -- mail_user 'ServerPassword' -- mail_password );
    IF result_code = 0 THEN
        SET result_code = xp_sendmail( 'IS Group', --recipient 'Backup', -- subject 'message'='Backup completed' -- message );
    END IF;
END;
```
ENDIF;
IF result_code = 0 THEN
    SET result_code = xp_stopmail();
ENDIF;
IF result_code = 0 THEN
    MESSAGE 'Backup completed message successfully sent';
ELSE
    SET error_code = xp_get_mail_error_code();
    SET error_text = xp_get_mail_error_text();
    MESSAGE 'Error: ' || result_code ||
    ' Code: ' || error_code ||
    ' Text: ' || error_text;
ENDIF;
END;

See also
- “xp_startmail system procedure” on page 1340
- “xp_startsmtp system procedure” on page 1341
- “xp_sendmail system procedure” on page 1336
- “xp_stopmail system procedure” on page 1343
- “xp_stopsmtp system procedure” on page 1344
- “xp_get_mail_error_code system procedure” on page 1329
- “xp_get_mail_error_text system procedure” on page 1331

Directory and file system procedures

You can access the local file structure of the computer running a database server by:

- Directory access servers

- File and directory system procedures, such as the sp_create_directory system procedure

The file and directory system procedures are easier to use than directory access servers, but they are not as flexible nor as powerful as directory access proxy tables and servers.

Use the following system procedures to manipulate directories and files on the computer where a server resides:

- **sp_list_directory system procedure** Lists the contents of a specified directory.

- **sp_create_directory system procedure** Creates a specified directory.

- **sp_copy_directory system procedure** Copies specified directory to another location.

- **sp_move_directory system procedure** Moves a specified directory.

- **sp_delete_directory system procedure** Deletes a specified directory.

- **sp_copy_file system procedure** Copies a specified file.

- **sp_move_file system procedure** Moves a specified file
sp_delete_file system procedure  Deletes a specified file.

The following system procedures can also be used to read and write to files:

xp_read_file system procedure  Reads a file and returns the contents of the file as a LONG BINARY variable.

xp_write_file system procedure  Writes data to a file from a SQL statement.

See also

- “sp_copy_directory system procedure” on page 1270
- “sp_copy_file system procedure” on page 1271
- “sp_create_directory system procedure” on page 1272
- “sp_delete_directory system procedure” on page 1275
- “sp_delete_file system procedure” on page 1275
- “sp_list_directory system procedure” on page 1286
- “sp_move_directory system procedure” on page 1289
- “sp_move_file system procedure” on page 1291
- “xp_read_file system procedure” on page 1334
- “xp_write_file system procedure” on page 1344
- “Directory access servers” [SQL Anywhere Server - SQL Usage]

Secure feature system procedures

The following system procedures are for use with secure features:

- “sa_server_option system procedure”

- “sp_alter_secure_feature_key system procedure”

- “sp_create_secure_feature_key system procedure”

- “sp_drop_secure_feature_key system procedure”

- “sp_list_secure_feature_keys system procedure”

- “sp_use_secure_feature_key system procedure”

Adaptive Server Enterprise system and catalog procedures

Adaptive Server Enterprise provides system and catalog procedures to carry out many administrative functions and to obtain system information. System procedures are built-in stored procedures used for getting reports from and updating system tables; catalog stored procedures retrieve information from the system tables in tabular form.
SQL Anywhere has implemented support for some of these Adaptive Server Enterprise procedures. However, for information about using these procedures, refer to your Adaptive Server Enterprise documentation.

**Adaptive Server Enterprise system procedures**

SQL Anywhere includes some Adaptive Server Enterprise system procedures. While these procedures perform the same functions as they do in Adaptive Server Enterprise, they are not identical. If you have preexisting scripts that use these procedures, you may want to review the procedure code. To see the text of a stored procedure, execute the following command.

```
sp_helpdata 'dbo.procedure_name'
```

The implemented system procedures are described in the following table.

<table>
<thead>
<tr>
<th>System procedure name / parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_addgroup @grpname</td>
<td>Adds a group to a database</td>
</tr>
<tr>
<td>sp_addlogin @login_name , @passwd [, @defaultdb [, @deflanguage [, @full-name ] ] ]</td>
<td>Adds a new login ID to a database</td>
</tr>
<tr>
<td>sp_addmessage @message_num , @message_text [, @language ]</td>
<td>Adds a user-defined message to ISYSUSERMESSAGE, for use by stored procedure PRINT and RAISERROR calls</td>
</tr>
<tr>
<td>sp_addtype @typename , @phystype [, @ident_null ]</td>
<td>Creates a user-defined data type</td>
</tr>
<tr>
<td>sp_adduser @login_name [, @name_in_db [, @grpname ] ]</td>
<td>Adds a new user ID to a database</td>
</tr>
<tr>
<td>sp_changeuser @ grpname , @name_in_db</td>
<td>Changes a user group or adds a user to a group</td>
</tr>
<tr>
<td>sp_dropgroup @grpname</td>
<td>Drops a group from a database</td>
</tr>
<tr>
<td>sp_droplogin @login_name</td>
<td>Drops a login ID from a database</td>
</tr>
<tr>
<td>sp_dropmessage @message_number [, @language ]</td>
<td>Drops a user-defined message</td>
</tr>
<tr>
<td>sp_droptype @typename</td>
<td>Drops a user-defined data type</td>
</tr>
<tr>
<td>sp_dropuser @name_in_db</td>
<td>Drops a user ID from a database</td>
</tr>
<tr>
<td>System procedure name / parameters</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>sp_getmessage @message_num , @msg_var [, @language]</td>
<td>Retrieves a stored message string from ISYSUSERMESS-SAGE, for PRINT and RAISERROR statements.</td>
</tr>
<tr>
<td>sp_helptext [ @objname ]</td>
<td>Displays the text of a system procedure, trigger, or view</td>
</tr>
<tr>
<td>sp_password @caller_pswd , @new_pswd [, @login_name ]</td>
<td>Adds or changes a password for a user ID</td>
</tr>
</tbody>
</table>

**Adaptive Server Enterprise catalog procedures**

SQL Anywhere implements a subset of the Adaptive Server Enterprise catalog procedures. While these procedures perform the same functions as they do in Adaptive Server Enterprise, they are not identical. If you have preexisting scripts that use these procedures, you may want to review the procedure code. To see the text of a stored procedure, execute the following command.

```
sp_helptext 'dbo.procedure_name'
```

The implemented catalog procedures are described in the following table.

<table>
<thead>
<tr>
<th>Catalog procedure name / parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_column_privileges</td>
<td>Unsupported</td>
</tr>
<tr>
<td>sp_columns [ @table_name [, @table_owner [, @table_qualifier [, @column_name ]]] ]</td>
<td>Returns the data types of the specified columns</td>
</tr>
<tr>
<td>sp_fkeys [ @pktable_name [, @pktable_owner [, @pktable_qualifier [, @fktable_name [, @fktable_owner [, @fktable_qualifier ]]]]] ]</td>
<td>Returns foreign key information about the specified table</td>
</tr>
<tr>
<td>sp_pkeys @table_name [, @table_owner [, @table_qualifier ] ]</td>
<td>Returns primary key information about the specified table</td>
</tr>
<tr>
<td>sp_special_columns @table_name [, @table_owner [, @table_qualifier [, @col_type ]]]</td>
<td>Returns the optimal set of columns that uniquely identify a row in the specified table</td>
</tr>
<tr>
<td>sp_sproc_columns @sp_name [, @sp_owner [, @sp_qualifier [, @column_name ]]]</td>
<td>Returns information about the input and return parameters of a stored procedure</td>
</tr>
<tr>
<td>sp_statistics [ @table_name [, @table_owner [, @table_qualifier [, @index_name [, @is_unique ]]]] ]</td>
<td>Returns information about tables and their indexes</td>
</tr>
<tr>
<td>sp_stored_procedures [ @sp_name [, @sp_owner [, @sp_qualifier ]]]</td>
<td>Returns information about one or more stored procedures</td>
</tr>
</tbody>
</table>
### Alphabetical list of system procedures

System procedures are owned by dbo. Some of these procedures are for internal system use. This section documents only those not intended solely for system and internal use.

#### `sa_ansi_standard_packages` system procedure

Returns information about the non-core SQL extensions used in a SQL statement.

**Syntax**

```sql
sa_ansi_standard_packages(
    standard
    , statement
)
```

**Arguments**

- **standard** Use this LONG VARCHAR parameter to specify the standard to use for the core extensions. One of SQL:1999 or SQL:2003.
- **statement** Use this LONG VARCHAR parameter to specify the SQL statement to evaluate.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>package_id</td>
<td>VARCHAR(10)</td>
<td>The feature identifier.</td>
</tr>
<tr>
<td>package_name</td>
<td>LONG VARCHAR</td>
<td>The feature name.</td>
</tr>
</tbody>
</table>

**Remarks**

If there are no non-core extensions used for the statement, the result set is empty.

**Privileges**

None

**Side effects**

None
See also

- “The SQL preprocessor” [SQL Anywhere Server - Programming]
- “SQLFLAGGER function [Miscellaneous]” on page 369
- “sql_flagger_error_level option” [SQL Anywhere Server - Database Administration]
- “sql_flagger_warning_level option” [SQL Anywhere Server - Database Administration]

Example

Following is an example call to the sa_ansi_standard_packages system procedure:

```sql
CALL sa_ansi_standard_packages( 'SQL:2003',
'SELECT *
  FROM ( SELECT o.SalesRepresentative,
         o.Region,
         SUM( s.Quantity * p.UnitPrice ) AS total_sales,
         DENSE_RANK() OVER ( PARTITION BY o.Region,
                             GROUPING( o.SalesRepresentative )
                     ORDER BY total_sales DESC ) AS sales_rank
FROM Product p, SalesOrderItems s, SalesOrders o
WHERE p.ID = s.ProductID AND s.ID = o.ID
GROUP BY GROUPING SETS( ( o.SalesRepresentative, o.Region ),
                          o.Region ) ) AS DT
WHERE sales_rank <= 3
ORDER BY Region, sales_rank');
```

The example generates the following result set:

<table>
<thead>
<tr>
<th>package_id</th>
<th>package_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>T612</td>
<td>Advanced OLAP operations</td>
</tr>
<tr>
<td>T611</td>
<td>Elementary OLAP operations</td>
</tr>
<tr>
<td>F591</td>
<td>Derived tables</td>
</tr>
<tr>
<td>T431</td>
<td>Extended grouping capabilities</td>
</tr>
</tbody>
</table>

**sa_audit_string system procedure**

Adds a string to the transaction log.

**Syntax**

```sql
sa_audit_string( string )
```

**Arguments**

- **string** The VARCHAR(128) string of characters to add to the transaction log.

**Remarks**

If auditing is turned on, this system procedure adds a comment to the auditing information stored in the transaction log. The string can be a maximum of 128 characters.
Privileges

You must have the MANAGE AUDITING system privilege.

Side effects

None

See also

- “auditing option” [SQL Anywhere Server - Database Administration]
- “Database activity audits” [SQL Anywhere Server - Database Administration]

Example

The following example uses sa_audit_string to add a comment to the transaction log:

```sql
CALL sa_audit_string( 'Auditing test' );
```

sa_certificate_info system procedure

Displays information about the specified certificate that is stored in the database.

Syntax

```sql
sa_certificate_info( cert_name )
```

Arguments

- **cert_name**

  The CHAR(128) certificate name that was used in the CREATE CERTIFICATE statement.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>CHAR(128)</td>
<td>The name of the attribute.</td>
</tr>
<tr>
<td>value</td>
<td>LONG VARCHAR</td>
<td>The value of the attribute.</td>
</tr>
</tbody>
</table>

Remarks

If cert_name is NULL or no certificate in the ISYSCERTIFICATE table has that name, no rows are returned. Otherwise, the following keys are always returned:

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Value of the cert_name column in ISYSCERTIFICATE.</td>
</tr>
<tr>
<td>ID</td>
<td>Value of the object_id column in ISYSCERTIFICATE.</td>
</tr>
</tbody>
</table>

Other rows returned have keys that correspond to attributes of the certificate such as Common Name, Country Code, Locality, and Organization.
Privileges

None

Side effects

None

See also

- “CREATE CERTIFICATE statement” on page 547
- “SYSCERTIFICATE system view” on page 1350

Example

The following example returns information about the certificate mycert:

```
CALL sa_certificate_info( 'mycert' );
```

**sa_char_terms system procedure**

Breaks a CHAR string into terms and returns each term as a row along with its position.

**Syntax**

```
sa_char_terms(
    text
    [, config_name
    [, owner ] ]
)
```

**Arguments**

- **text**  The LONG VARCHAR string you are parsing.

- **config_name**  Use this optional CHAR(128) parameter to specify the text configuration object to apply when processing the string. The default value is 'default_char'.

- **owner**  Use this optional CHAR(128) parameter to specify the owner of the text configuration object. The default value is NULL. The current user is assumed if the owner is not specified or if NULL is specified.

**Remarks**

You can use this system procedure to find out how a string is interpreted when the settings for a text configuration object are applied. This can be helpful when you want to know what terms would be dropped during indexing or from a query string.

Privileges

None

Side effects

None
See also

- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Text configuration object concepts and reference” [SQL Anywhere Server - SQL Usage]
- “sa_nchar_terms system procedure” on page 1195

Example

The following statement returns the terms in the CHAR string "It's a work-at-home day!" using the default CHAR text configuration object, default_char:

```sql
CALL sa_char_terms ('It''s a work-at-home day!', 'default_char', 'sys');
```

<table>
<thead>
<tr>
<th>term</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>It</td>
<td>1</td>
</tr>
<tr>
<td>s</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>work</td>
<td>4</td>
</tr>
<tr>
<td>at</td>
<td>5</td>
</tr>
<tr>
<td>home</td>
<td>6</td>
</tr>
<tr>
<td>day</td>
<td>7</td>
</tr>
</tbody>
</table>

**sa_check_commit system procedure**

Checks for outstanding referential integrity violations before a commit.

**Syntax**

```sql
sa_check_commit(    
    tname
    , keyname
)
```

**Arguments**

- **tname** A VARCHAR(128) parameter containing the name of a table with a row that is currently violating referential integrity.

- **keyname** A VARCHAR(128) parameter containing the name of the corresponding foreign key index.
Remarks
If the database option wait_for_commit is On, or if a foreign key is defined using CHECK ON COMMIT in the CREATE TABLE statement, you can update the database and cause a referential integrity violation if the violations are resolved before the changes are committed.

You can use the sa_check_commit system procedure to check whether there are any outstanding referential integrity violations before attempting to commit your changes.

The returned parameters indicate the name of a table containing a row that is currently violating referential integrity, and the name of the corresponding foreign key index.

Privileges
None

Side effects
None

See also
● “wait_for_commit option” [SQL Anywhere Server - Database Administration]
● “CREATE TABLE statement” on page 690

Example
The following set of statements can be executed from Interactive SQL. Rows are deleted from the Departments table in the sample database and a referential integrity violation occurs. The call to the sa_check_commit system procedure checks which tables and keys have outstanding violations, and the rollback cancels the change.

```
SET TEMPORARY OPTION wait_for_commit='On';
DELETE FROM Departments;
CREATE OR REPLACE VARIABLE tname VARCHAR(128);
CREATE OR REPLACE VARIABLE keyname VARCHAR(128);
CALL sa_check_commit(tname, keyname);
SELECT tname, keyname;
ROLLBACK;
SET TEMPORARY OPTION wait_for_commit='Off';
```

sa_clean_database system procedure
Starts the database cleaner and sets the maximum length of time for which it can run.

Syntax
```
sa_clean_database([duration])
```

Arguments
● duration Use this optional UNSIGNED INTEGER parameter to specify the number of seconds that the clean operation is allowed to run. The default is 0 which is interpreted to mean that no limit is imposed on the duration that the cleaner runs.
Remarks

The database cleaner is an internal task that runs on a default schedule. You can use this system procedure to force the database cleaner to run immediately and to specify how long the cleaner can run each time it is invoked. When the duration is 0, the database cleaner runs until all pages in all dbspaces have been cleaned.

If you use this system procedure to start the database cleaner while a database is being validated, the database cleaner does not run until validation is complete.

Some database tasks, such as processing snapshot isolation transactions, index maintenance, and deleting rows, can execute more efficiently if some portions of the request are deferred to a later time. These deferrable activities typically involve cleanup by removing deleted, historical, and otherwise unnecessary entries from database pages, or reorganizing database pages for more efficient access.

Postponing some of these activities not only allows the current request to finish more quickly, it potentially allows cleanup to occur when the database server is less active. These unnecessary entries are identified so that they are not visible to other transactions; however, they do take up space on a page, and must be removed at some point.

The database cleaner performs any deferred cleanup activities. It is scheduled to run every 20 seconds. When it is invoked, the database cycles sequentially through the database's dbspaces, examining and cleaning each cleanable page before moving on to the next one. When invoked automatically by the database server, the database cleaner is a self-tuning process. The amount of work that the database cleaner performs, and the duration for which it executes, depend on several factors, including the fraction of outstanding cleanable pages in a dbspace, the current amount of activity in the database server, and the amount of time that the database cleaner has already spent cleaning. If, after running for 0.5 seconds, the cleaner detects active requests in the server, it stops and reschedules itself to execute at its regular interval. The database cleaner attempts to process pages when there are no other requests executing in the server, and therefore takes advantage of periods of server inactivity.

Database cleaner statistics are available through four database properties:

- **CleanablePagesAdded** returns the number of pages that need to be cleaned
- **CleanablePagesCleaned** returns the number of pages that have already been cleaned
- **CleanableRowsAdded** returns the number of rows that need to be cleaned
- **CleanableRowsCleaned** returns the number of rows that have already been cleaned

The difference between the values of CleanablePagesAdded and CleanablePagesCleaned indicates how many database pages still require cleaning.

You can use the sa_clean_database system procedure to configure the database cleaner to run until all the pages in a database are cleaned, or to specify a maximum duration for the database cleaner to run.

To further customize the behavior of the database cleaner, you can set up an event that starts the database cleaner if the number of pages or rows that need to be cleaned exceed a specified threshold.
Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

None

See also

- “CREATE EVENT statement” on page 570
- “CleanablePagesAdded database property” [SQL Anywhere Server - Database Administration]
- “CleanablePagesCleaned database property” [SQL Anywhere Server - Database Administration]
- “CleanableRowsAdded database property” [SQL Anywhere Server - Database Administration]
- “CleanableRowsCleaned database property” [SQL Anywhere Server - Database Administration]

Example

The following example sets the duration of the database cleaner to 10 seconds:

```
CALL sa_clean_database( 10 );
```

The following example creates a scheduled event that runs daily to allow the database cleaner to run until all pages in the database are cleaned:

```
CREATE EVENT DailyDatabaseCleanup
SCHEDULE
START TIME '6:00 pm'
ON ( 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday' )
HANDLER
BEGIN
  CALL sa_clean_database( );
END;
```

The following example forces the database cleaner to run when 20% or more of the pages in the database need to be cleaned:

```
CREATE EVENT PeriodicCleaner
SCHEDULE
BETWEEN '9:00 am' and '5:00 pm'
EVERY 1 HOURS
HANDLER
BEGIN
  DECLARE @num_db_pages INTEGER;
  DECLARE @num_dirty_pages INTEGER;
  -- Get the number of database pages
  SELECT (SUM( DB_EXTENDED_PROPERTY( 'FileSize', t.dbspace_id ) -
             DB_EXTENDED_PROPERTY( 'FreePages', t.dbspace_id ) ))
  INTO @num_db_pages
  FROM (SELECT dbspace_id FROM SYSDBSPACE) AS t;
  -- Get the number of dirty pages to be cleaned
  SELECT (DB_PROPERTY( 'CleanablePagesAdded' ) -
          DB_PROPERTY( 'CleanablePagesCleaned' ))
  INTO @num_dirty_pages;
  -- Check whether the number of dirty pages exceeds 20% of
  -- the size of the database
  IF @num_dirty_pages > @num_db_pages * 0.20 THEN
    CALL sa_clean_database( );
  END;
END;
```
sa_column_stats system procedure

Returns various statistics about the specified column(s). These statistics are not related to the column statistics maintained for use by the optimizer.

Syntax

```
sa_column_stats(  
    tab_name  
    [ , col_name  
    [ , tab_owner  
    [ , max_rows ]] ]  
)
```

Arguments

- **tab_name** This optional CHAR(128) parameter specifies the name of the table. If this parameter is not specified, statistics are calculated for all columns in all table(s). The default is '%'.

- **col_name** This optional CHAR(128) parameter specifies the columns for which to calculate statistics. If this parameter is not specified, statistics are calculated for all columns in the specified table(s). The default is '%'.

- **tab_owner** This optional CHAR(128) parameter specifies the owner of the table. If this parameter is not specified, the database server uses the owner of the first table that matches the `tab_name` specified. The default is '%'.

- **max_rows** This optional INTEGER parameter specifies the number of rows to use for the calculations. The default is 1000. Specifying 0 instructs the database server to calculate the ratio based on all the rows in the table.

Result set

With the exception of table_owner, table_name, and column_name, all values in the result set are NULL for non-string columns. Also, for empty tables, num_rows_processed and num_values_compressed are 0, while all other values are NULL.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_owner</td>
<td>CHAR(128)</td>
<td>The owner of the table.</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>The table name.</td>
</tr>
<tr>
<td>column_name</td>
<td>CHAR(128)</td>
<td>The column name.</td>
</tr>
<tr>
<td>num_rows_processed</td>
<td>INTEGER</td>
<td>The total number of rows read to calculate the statistics.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>num_values_compressed</td>
<td>INTEGER</td>
<td>The number of values in the column that are compressed. If the column is not compressed, the value is 0.</td>
</tr>
<tr>
<td>avg_compression_ratio</td>
<td>DOUBLE</td>
<td>The average compression ratio, expressed as a percentage reduction in size, for compressed values in the column. If the column is not compressed, the value is NULL.</td>
</tr>
<tr>
<td>avg_length</td>
<td>DOUBLE</td>
<td>The average length of all non-NULL strings in the column.</td>
</tr>
<tr>
<td>stddev_length</td>
<td>DOUBLE</td>
<td>The standard deviation of the lengths of all non-NULL strings in the column.</td>
</tr>
<tr>
<td>min_length</td>
<td>INTEGER</td>
<td>The minimum length of non-NULL strings in the column.</td>
</tr>
<tr>
<td>max_length</td>
<td>INTEGER</td>
<td>The maximum length of strings in the column.</td>
</tr>
<tr>
<td>avg_uncompressed_length</td>
<td>DOUBLE</td>
<td>The average length of all uncompressed, non-NULL strings in the column.</td>
</tr>
<tr>
<td>stddev_uncompressed_length</td>
<td>DOUBLE</td>
<td>The standard deviation of the lengths of all uncompressed, non-NULL strings in the column.</td>
</tr>
<tr>
<td>min_uncompressed_length</td>
<td>INTEGER</td>
<td>The minimum length of all uncompressed, non-NULL strings in the column.</td>
</tr>
<tr>
<td>max_uncompressed_length</td>
<td>INTEGER</td>
<td>The maximum length of all uncompressed, non-NULL strings in the column.</td>
</tr>
</tbody>
</table>

**Remarks**

The database server determines the columns that match the owner, table, and column names specified, and then for each one, calculates statistics for the data in each specified column. By default, the database server only uses the first 1000 rows of data.

For `avg_compression_ratio`, values cannot be greater than, or equal to 100, however, they can be less than 0 if highly incompressible data (for example, data that is already compressed) is inserted into a compressed column. Higher values indicate better compression. For example, if the number returned is 80, then the size of the compressed data is 80% less than the size of the uncompressed data.

**Privileges**

You must be the owner of table, or have SELECT privilege on the columns, or have the MONITOR or MANAGE ANY STATISTICS system privilege.

**Side effects**

None
See also

- “Column compression considerations” [SQL Anywhere Server - Database Administration]

Example

In this example, you use the `sa_column_stats` system procedure in a SELECT statement to determine which columns in the database are benefiting most from column compression:

```sql
SELECT * FROM sa_column_stats()
WHERE num_values_compressed > 0
ORDER BY avg_compression_ratio desc;
```

In this example, you narrow your selection from the previous example to tables owned by bsmith:

```sql
SELECT * FROM sa_column_stats( tab_owner='GROUPO' )
WHERE num_values_compressed > 0
ORDER BY avg_compression_ratio desc;
```

**sa_conn_activity system procedure**

Returns the most recently-prepared SQL statement for each connection to the indicated database on the server.

**Syntax**

```sql
sa_conn_activity( [ connidparm ] )
```

**Arguments**

- `connidparm` Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>INTEGER</td>
<td>Returns the connection ID (a number) for the current connection.</td>
</tr>
<tr>
<td>Name</td>
<td>VARCHAR(255)</td>
<td>Returns the name of the current connection.</td>
</tr>
<tr>
<td>Userid</td>
<td>VARCHAR(255)</td>
<td>Returns the user ID for the connection.</td>
</tr>
<tr>
<td>DBNumber</td>
<td>INTEGER</td>
<td>Returns the ID number of the database.</td>
</tr>
<tr>
<td>LastReqTime</td>
<td>VARCHAR(255)</td>
<td>Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.</td>
</tr>
</tbody>
</table>
### LastStatement

**Data type**: LONG VARCHAR

**Description**: Returns the most recently prepared SQL statement for the current connection.

### Remarks

If `connidparm` is less than zero, then information for the current connection is returned. If `connidparm` is not supplied or is NULL, then information is returned for all connections to all databases running on the database server.

The `sa_conn_activity` system procedure returns a result set consisting of the most recently-prepared SQL statement for the connection. Recording of statements must be enabled for the database server before calling `sa_conn_activity`. To do this, specify the `-zl` option when starting the database server, or execute the following:

```sql
CALL sa_server_option('RememberLastStatement','ON');
```

This procedure is useful when the database server is busy and you want to obtain information about the last SQL statement prepared for each connection. This feature can be used as an alternative to request logging.

*Cloud note*: Because of tenant database isolation rules, when this system procedure is run in the cloud it returns only information about the current tenant database.

### Privileges

No privileges are required to execute this system procedure for the current connection ID. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

### Side effects

None

### See also

- “-zl database server option” [SQL Anywhere Server - Database Administration]
- “sa_server_option system procedure” on page 1222
- “Name connection property” [SQL Anywhere Server - Database Administration]
- “List of connection properties” [SQL Anywhere Server - Database Administration]

### Example

The following example uses the `sa_conn_activity` system procedure to display the most recently-prepared SQL statement for each connection.

```sql
CALL sa_conn_activity( );
```

| Number | Name                | Userid | DBNumber | ...
|--------|---------------------|--------|----------|------
| 1,949  | SQL_DBC_117acc40    | DBA    | 0        | ...  |
sa_conn_compression_info system procedure

Summarizes communication compression rates.

Syntax

```
sa_conn_compression_info( [ connidparm ] )
```

Arguments

- **connidparm** Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>VARCHAR(20)</td>
<td>Returns a string identifying whether the compression statistics that follow represent either one connection (Connection), or all connections to the server (Server).</td>
</tr>
<tr>
<td>ConnNumber</td>
<td>INTEGER</td>
<td>Returns an INTEGER representing a connection ID number. Returns NULL if the Type is Server.</td>
</tr>
<tr>
<td>Compression</td>
<td>VARCHAR(10)</td>
<td>Returns On or Off to indicate whether communication compression is enabled on the connection. Returns NULL if the Type is Server, or ON/OFF if the Type is Connection.</td>
</tr>
<tr>
<td>TotalBytes</td>
<td>INTEGER</td>
<td>Returns an INTEGER representing the total number of actual bytes both sent and received.</td>
</tr>
<tr>
<td>TotalBytesUn-Comp</td>
<td>INTEGER</td>
<td>Returns an INTEGER representing the number of bytes that would have been sent and received if compression were disabled.</td>
</tr>
<tr>
<td>CompRate</td>
<td>NUMERIC(5,2)</td>
<td>Returns a NUMERIC (5,2) value representing the overall compression rate. For example, a value of 0 indicates that no compression occurred. A value of 75 indicates that the data was compressed by 75%, or down to one quarter of its original size.</td>
</tr>
<tr>
<td>CompRateSent</td>
<td>NUMERIC(5,2)</td>
<td>Returns a NUMERIC (5,2) value representing the compression rate for data sent to the client.</td>
</tr>
</tbody>
</table>
### Column name | Data type | Description
--- | --- | ---
CompRateReceived | NUMERIC(5,2) | Returns a NUMERIC (5,2) value representing the compression rate for data received from the client.
TotalPackets | INTEGER | Returns an INTEGER representing the total number of actual packets both sent and received.
TotalPacketsUnComp | INTEGER | Returns an INTEGER representing the total number of packets that would have been sent and received if compression was disabled.
CompPktRate | NUMERIC(5,2) | Returns a NUMERIC (5,2) value representing the overall compression rate of packets.
CompPktRateSent | NUMERIC(5,2) | Returns a NUMERIC (5,2) value representing the compression rate of packets sent to the client.
CompPktRateReceived | NUMERIC(5,2) | Returns a NUMERIC (5,2) value representing the compression rate of packets received from the client.

**Remarks**

If `connidparm` is less than zero, then a result set consisting of compression properties for the current connection is returned. If `connidparm` is not supplied or is NULL, then compression properties are returned for all connections to all databases running on the database server.

**Privileges**

No privileges are required to execute this system procedure for the current connection ID. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

**Side effects**

None

**See also**

- “List of connection properties” [SQL Anywhere Server - Database Administration]

**Example**

The following example uses the `sa_conn_compression_info` system procedure to return a result set summarizing compression properties for all connections to the server.

```sql
CALL sa_conn_compression_info( );
```

| Type    | ConnNumber | Compression | TotalBytes | ...
|---------|------------|-------------|------------|---
| Connection | 79         | Off         | 7841       | ... |
### System procedures

| Type   | ConnNumber | Compression | TotalBytes | ...
|--------|------------|-------------|------------|---
| Server | (NULL)     | (NULL)      | 2737761    | ...
| ...    | ...        | ...         | ...        | ...

**sa_conn_info system procedure**

Reports connection property information.

**Syntax**

```sql
sa_conn_info( [ connidparm ] )
```

**Arguments**

- `connidparm` This optional INTEGER parameter specifies the connection ID number. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>INTEGER</td>
<td>Returns the connection ID (a number) for the current connection.</td>
</tr>
<tr>
<td>Name</td>
<td>VAR-CHAR(255)</td>
<td>Returns the connection ID (a number) for the current connection. Temporary connection names have INT: prepended to the connection name.</td>
</tr>
<tr>
<td>Userid</td>
<td>VAR-CHAR(255)</td>
<td>Returns the user ID for the connection.</td>
</tr>
<tr>
<td>DBNumber</td>
<td>INTEGER</td>
<td>Returns the ID number of the database.</td>
</tr>
<tr>
<td>LastReq-Time</td>
<td>VAR-CHAR(255)</td>
<td>Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.</td>
</tr>
<tr>
<td>ReqType</td>
<td>VAR-CHAR(255)</td>
<td>Returns the type of the last request. If a connection has been cached by connection pooling, its ReqType value is CON-NECT_POOL_CACHE.</td>
</tr>
<tr>
<td>CommLink</td>
<td>VAR-CHAR(255)</td>
<td>Returns the communication link for the connection. This is one of the network protocols supported by SQL Anywhere, or local for a same-computer connection.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NodeAddr</td>
<td>VAR-CHAR(255)</td>
<td>Returns the address of the client in a client/server connection.</td>
</tr>
<tr>
<td>ClientPort</td>
<td>INTEGER</td>
<td>Returns the client's TCP/IP port number or 0 if the connection isn't a TCP/IP connection.</td>
</tr>
<tr>
<td>ServerPort</td>
<td>INTEGER</td>
<td>Returns the database server's TCP/IP port number or 0.</td>
</tr>
<tr>
<td>BlockedOn</td>
<td>INTEGER</td>
<td>Returns zero if the current connection isn't blocked, or if it is blocked, the connection number on which the connection is blocked because of a locking conflict.</td>
</tr>
<tr>
<td>LockRowID</td>
<td>UNSIGNED BIGINT</td>
<td>Returns the identifier of the locked row.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LockRowID is NULL if the connection is not waiting on a lock associated with a row (that is, it is not waiting on a lock, or it is waiting on a lock that has no associated row).</td>
</tr>
<tr>
<td>LockIndexID</td>
<td>INTEGER</td>
<td>Returns the identifier of the locked index.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LockIndexID is -1 if the lock is associated with all indexes on the table in LockTable. LockIndexID is NULL if the connection is not waiting on a lock associated with an index (that is, it is not waiting on a lock, or it is waiting on a lock that has no associated index).</td>
</tr>
<tr>
<td>LockTable</td>
<td>VAR-CHAR(255)</td>
<td>Returns the name of the table associated with a lock if the connection is currently waiting for a lock. Otherwise, LockTable returns an empty string.</td>
</tr>
<tr>
<td>UncommittOps</td>
<td>INTEGER</td>
<td>Returns the number of uncommitted operations.</td>
</tr>
<tr>
<td>ParentConnection</td>
<td>INTEGER</td>
<td>Returns the connection ID of the connection that created a temporary connection to perform a database operation (such as performing a backup or creating a database). For other types of connections, this property returns NULL.</td>
</tr>
</tbody>
</table>

**Remarks**

If `connidparm` is less than zero, then a result set consisting of connection properties for the current connection is returned. If `connidparm` is not supplied or is NULL, then connection properties are returned for all connections to all databases running on the database server.

In a block situation, the BlockedOn value returned by this procedure allows you to check which users are blocked, and who they are blocked on. The `sa_locks` system procedure can be used to display the locks held by the blocking connection.
For more information based on any of these properties, you can execute something similar to the following:

```
SELECT *, DB_NAME( DBNumber ),
       CONNECTION_PROPERTY( 'LastStatement', Number )
FROM sa_conn_info( );
```

The value of LockRowID can be used to look up a lock in the output of the `sa_locks` procedure.

The value in LockIndexID can be used to look up a lock in the output of the `sa_locks` procedure. Also, the value in LockIndexID corresponds to the primary key of the ISYSIDX system table, which can be viewed using the SYSIDX system view.

Every lock has an associated table, so the value of LockTable can be used to unambiguously determine whether a connection is waiting on a lock.

*Cloud note:* Because of tenant database isolation rules, when this system procedure is run in the cloud it returns only information about the current tenant database.

**Privileges**

No privileges are required to execute this system procedure for the current connection ID. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

**Side effects**

None

**See also**

- “List of connection properties” [*SQL Anywhere Server - Database Administration*]
- “Name connection property” [*SQL Anywhere Server - Database Administration*]
- “sa_locks system procedure” on page 1168
- “SYSIDX system view” on page 1364

**Examples**

The following example uses the `sa_conn_info` system procedure to return a result set summarizing connection properties for all connections to the server.

```
CALL sa_conn_info( );
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Userid</th>
<th>DBNumber</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>SQL_DBC_10dcf810</td>
<td>DBA</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>setup</td>
<td>User1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The following example uses the `sa_conn_info` system procedure to return a result set showing which connection created a temporary connection.
SELECT Number, Name, ParentConnection FROM sa_conn_info();

Connection 8 created the temporary connection that executed a CREATE DATABASE statement.

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>ParentConnection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000048</td>
<td>INT: CreateDB</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>SQL_DBC_14675af8</td>
<td>(NULL)</td>
</tr>
<tr>
<td>8</td>
<td>SQL_DBA_152d5ac0</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

**sa_conn_list system procedure**

Returns a result set containing connection IDs.

**Syntax**

```sql
sa_conn_list(
   [ connidparm
   [, dbidparm ]]
)
```

**Arguments**

- **connidparm** Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.
- **dbidparm** Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>INTEGER</td>
<td>Returns the connection ID (a number) for the current connection.</td>
</tr>
</tbody>
</table>

**Remarks**

If `connidparm` is greater than zero, then information for the supplied connection is returned. If `connidparm` is less than zero, then information for the current connection is returned. If `connidparm` and `dbidparm` are not supplied or are NULL, then connection IDs for all connections to all databases running on the database server are returned.

If `connidparm` is NULL and `dbidparm` is greater than or equal to zero, then connection IDs for only that database are returned. If `connidparm` is NULL and `dbidparm` is less than zero, then connection IDs for just the current database are returned.

**Privileges**

No privileges are required to execute this system procedure and return the current connection. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.
Side effects
None

See also
- “sa_db_list system procedure” on page 1119
- “sa_conn_options system procedure” on page 1110

Example
The following example uses the sa_conn_list system procedure to display a list of connection IDs.

```sql
CALL sa_conn_list( );
```

<table>
<thead>
<tr>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,949</td>
</tr>
<tr>
<td>1,948</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

sa_conn_options system procedure
Returns property information for connection properties that correspond to database options.

Syntax
```sql
sa_conn_options([ connidparm ])
```

Arguments
- `connidparm` Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>INTEGER</td>
<td>Returns the connection ID (a number) for the current connection.</td>
</tr>
<tr>
<td>PropNum</td>
<td>INTEGER</td>
<td>Returns the connection property number.</td>
</tr>
<tr>
<td>OptionName</td>
<td>VARCHAR(255)</td>
<td>Returns the option name.</td>
</tr>
<tr>
<td>OptionDescription</td>
<td>VARCHAR(255)</td>
<td>Returns the option description.</td>
</tr>
<tr>
<td>Value</td>
<td>LONG VARCHAR</td>
<td>Returns the option value.</td>
</tr>
</tbody>
</table>
Remarks
Returns the connection ID as Number, and the PropNum, OptionName, OptionDescription, and Value for each available connection property that corresponds to a database option.

If connidparm is less than zero, then option values for the current connection are returned. If connidparm is not supplied or is NULL, then option values are returned for all connections to the current database.

Privileges
No privileges are required to execute this system procedure for the current connection ID. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side effects
None

See also
● “sa_db_list system procedure” on page 1119
● “sa_conn_list system procedure” on page 1109
● “List of connection properties” [SQL Anywhere Server - Database Administration]
● “Database options” [SQL Anywhere Server - Database Administration]

Example
The following example uses the sa_conn_options system procedure to display property information for connection properties that correspond to database options.

```sql
CALL sa_conn_options( );
```

<table>
<thead>
<tr>
<th>Number</th>
<th>PropNum</th>
<th>OptionName</th>
<th>OptionDescription</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,952</td>
<td>461</td>
<td>blocking</td>
<td>Controls response to locking conflicts</td>
<td>On</td>
</tr>
<tr>
<td>1,952</td>
<td>462</td>
<td>blocking_timeout</td>
<td>Controls the time a transaction waits to obtain a lock</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

sa_conn_properties system procedure
Reports connection property information.

Syntax
```sql
sa_conn_properties( [ connidparm ] )
```

Arguments
- **connidparm** Use this optional INTEGER parameter to specify the connection ID number. The default is NULL.
Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>INTEGER</td>
<td>Returns the connection ID (a number) for the current connection.</td>
</tr>
<tr>
<td>PropNum</td>
<td>INTEGER</td>
<td>Returns the connection property number.</td>
</tr>
<tr>
<td>PropName</td>
<td>VARCHAR(255)</td>
<td>Returns the connection property name.</td>
</tr>
<tr>
<td>PropDescription</td>
<td>VARCHAR(255)</td>
<td>Returns the connection property description.</td>
</tr>
<tr>
<td>Value</td>
<td>LONG VARCHAR</td>
<td>Returns the connection property value.</td>
</tr>
</tbody>
</table>

Remarks

Returns the connection ID as Number, and the PropNum, PropName, PropDescription, and Value for each available connection property. Values are returned for all connection properties, database option settings related to connections, and statistics related to connections. Valid properties with NULL values are also returned.

If `connidparm` is less than zero, then property values for the current connection are returned. If `connidparm` is not supplied or is NULL, then property values are returned for all connections to the current database.

⚠ Cloud note: Because of tenant database isolation rules, when this system procedure is run in the cloud it returns only information about the current tenant database.

Privileges

No privileges are required to execute this system procedure for the current connection ID. To execute this system procedure for other connections, you must have either the SERVER OPERATOR, MONITOR, or DROP CONNECTION system privilege.

Side effects

None

See also

- “sa_conn_list system procedure” on page 1109
- “sa_conn_options system procedure” on page 1110
- “System functions” on page 154
- “List of connection properties” [SQL Anywhere Server - Database Administration]

Examples

The following example uses the `sa_conn_properties` system procedure to return a result set summarizing connection property information for all connections.

```sql
CALL sa_conn_properties();
```
This example uses the `sa_conn_properties` system procedure to return a list of all connections, in decreasing order by CPU time*:

```sql
SELECT Number AS connection_number,
       CONNECTION_PROPERTY ( 'Name', Number ) AS connection_name,
       CONNECTION_PROPERTY ( 'Userid', Number ) AS user_id,
       CAST ( Value AS NUMERIC ( 30, 2 ) ) AS approx_cpu_time
FROM sa_conn_properties()
WHERE PropName = 'ApproximateCPUTime'
ORDER BY approx_cpu_time DESC;
```

*Example courtesy of Breck Carter, RisingRoad Professional Services (http://www.risingroad.com).

### `sa_convert_ml_progress_to_timestamp` system procedure

For MobiLink scripted uploads only. This converts the progress value for scripted upload from an UNSIGNED BIGINT to a TIMESTAMP.

**Syntax**

```sql
sa_convert_ml_progress_to_timestamp( progress )
```

**Arguments**

- `progress` Use this UNSIGNED BIGINT parameter to specify the progress value to convert to a TIMESTAMP.

**Returns**

The function returns the TIMESTAMP that is represented by the value passed in.

**Remarks**

This function is the inverse of `sa_convert_timestamp_to_ml_progress`.

**Privileges**

None

**Side effects**

None
See also

- “sa_convert_timestamp_to_ml_progress system procedure” on page 1114
- “Scripted upload” [MobiLink - Client Administration]

Example

The following example converts an UNSIGNED BIGINT value to a timestamp value (2009-10-20 13:36:51.199).

```sql
SELECT sa_convert_ml_progress_to_timestamp( 3465034611199 );
```

**sa_convert_timestamp_to_ml_progress system procedure**

For MobiLink scripted uploads only. This converts the progress value for scripted upload from a TIMESTAMP to an UNSIGNED BIGINT.

**Syntax**

```sql
sa_convert_timestamp_to_ml_progress( t1 )
```

**Arguments**

- **t1** Use this TIMESTAMP parameter to specify the progress value to convert to an UNSIGNED BIGINT.

**Returns**

The function returns an UNSIGNED BIGINT that represents the timestamp passed in as a parameter.

**Remarks**

This procedure is the inverse of sa_convert_ml_progress_to_timestamp.

**Privileges**

None

**Side effects**

None

See also

- “sa_convert_ml_progress_to_timestamp system procedure” on page 1113
- “Scripted upload” [MobiLink - Client Administration]

**Examples**

The following examples convert timestamp values to UNSIGNED BIGINT values.

```sql
SELECT sa_convert_timestamp_to_ml_progress( CURRENT_TIMESTAMP );
SELECT sa_convert_timestamp_to_ml_progress( '2009-10-20 13:36:51.199' );
```
**sa_copy_cursor_to_temp_table system procedure**

Creates a temporary table and copies the result set of an open cursor to it.

**Syntax**

```sql
sa_copy_cursor_to_temp_table(
    cursor_name,
    table_name
[, first_row ]
[, max_rows ] ]
)
```

**Arguments**

- **cursor_name** Use this VARCHAR(256) parameter to specify the name of the open cursor.
- **table_name** Use this VARCHAR(256) parameter to specify the name of the temporary table.
- **first_row** Use this BIGINT parameter to specify the number of the first row to copy to the temporary table. The default is 1.
- **max_rows** Use this BIGINT parameter to specify the maximum number of rows to copy to the temporary table. The default is 9223372036854775807 (all rows).

**Remarks**

Suppose you have a cursor of several integer columns. `sa_copy_cursor_to_temp_table` creates a temporary table using a statement in this form:

```sql
BEGIN
    CREATE LOCAL TEMPORARY TABLE TempTab (
        col1 INT,
        col2 INT,
    ... rownum bigint primary key
    )
END;
```

`sa_copy_cursor_to_temp_table` names the columns col1,col2, and so on to avoid duplication of names or difficulty if cursor columns do not have a well defined name (for example, if they are a complex expression).

Once the temporary table is created, the contents of the open cursor are inserted by moving to the row number indicated by `first_row`, and inserting the number of rows indicated by `max_rows`. After the contents have been inserted into the temporary table, the cursor is re-positioned at its original location.

**Privileges**

None

**Side effects**

Copying from the cursor fetches the rows using the cursor’s isolation settings. This may acquire locks on rows and have other effects equivalent to fetching from the cursor.

If concurrent changes are made outside of the current connection and the cursor is not protected from these by materialization or isolation settings, then it is possible that the cursor will be positioned on a
different row after the procedure completes. For example, if the previous current row of the cursor was deleted, the cursor could be repositioned on the row after the original position.

If an error occurs while copying from the cursor, the cursor enters an invalid state, and further operations on the cursor fail with an error.

See also
- “sa_list.Cursors system procedure” on page 1166
- “sa_describe_cursor system procedure” on page 1124

Example
The following batch creates a cursor named myCursor and loads it with data from the Products table. The cursor is then opened (OPEN statement). A DROP statement drops myTempTable, if it already exists. Calling sa_copy_cursor_to_temp_table creates a temporary table called myTempTable and copies the contents of myCursor into it. Finally, a SELECT statement returns the data that was copied into the temporary table from the cursor:

```sql
BEGIN
  DECLARE myCursor CURSOR FOR
    SELECT ID, Name, Description, Color, Quantity FROM Products;
  OPEN myCursor;
  DROP TABLE IF EXISTS myTempTable;
  CALL sa_copy_cursor_to_temp_table( 'myCursor', 'myTempTable' );
  CLOSE myCursor;
  SELECT * FROM myTempTable;
END
```

**sa_cpu_topology system procedure**

[This topic has been updated for build 1823.]

Returns the processor topology of the computer that the database server is running on.

**Syntax**
```
sa_cpu_topology()
```

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>os_id</td>
<td>UNSIGNED INTEGER</td>
<td>Returns the logical processor identifier used by the underlying operating system. For example, on Windows this value matches the processor ID that appears in the Windows Task Manager.</td>
</tr>
<tr>
<td>socket</td>
<td>UNSIGNED INTEGER</td>
<td>Returns a CPU socket or package identifier.</td>
</tr>
<tr>
<td><strong>Column name</strong></td>
<td><strong>Data type</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>----------------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>core</td>
<td>UNSIGNED INTEGER</td>
<td>Returns an identifier for a core within a specific socket.</td>
</tr>
<tr>
<td>thread</td>
<td>UNSIGNED INTEGER</td>
<td>Returns an identifier for a thread within a specific socket and core.</td>
</tr>
<tr>
<td>apic</td>
<td>UNSIGNED INTEGER</td>
<td>Returns an identifier for the logical processor within the system. It contains the encoding of the socket, core, and thread.</td>
</tr>
<tr>
<td>&quot;group&quot;</td>
<td>UNSIGNED INTEGER</td>
<td>Returns the processor group number as assigned by the operating system on Windows. On Solaris, it is the processor set identifier. Otherwise, it is 0.</td>
</tr>
<tr>
<td>numa_node</td>
<td>UNSIGNED INTEGER</td>
<td>Returns the NUMA node assigned by the operating system on Windows.</td>
</tr>
<tr>
<td>online</td>
<td>BIT</td>
<td>Returns 1 if and only if the processor is online.</td>
</tr>
<tr>
<td>in_use</td>
<td>BIT</td>
<td>Returns 1 if the logical processor described by the current row is enabled in the process affinity mask for the SQL Anywhere database server process, and 0 otherwise.</td>
</tr>
</tbody>
</table>

**Remarks**

On all 32-bit and 64-bit Intel x86/x64 platforms (except for Mac versions where processor affinity cannot be controlled), the information returned by this procedure accurately describes the underlying hardware. On all other platforms, there is one row per logical processor in the system, but the column values are set as follows: os_id, socket, and apic are equal; core and thread are zero; core thread, and numa_node are zero, and group is also zero except on Solaris where it is the pset identifier. On some VM implementations, the results may not be accurate.

**Privileges**

None

**Side effects**

None

**See also**

- “-gta database server option” [SQL Anywhere Server - Database Administration]
- “sa_server_option system procedure” on page 1222

**Example**

The following example returns a result set containing the processor topology of the computer that the database server is running on.
CALL sa_cpu_topology( );

**sa_db_info system procedure**

Reports database property information.

**Syntax**

```
sa_db_info( [ dbidparm ] )
```

**Arguments**

- **dbidparm**  
  Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>INTEGER</td>
<td>Returns the connection ID (a number) for the current connection.</td>
</tr>
<tr>
<td>Alias</td>
<td>VARCHAR(255)</td>
<td>Returns the database name.</td>
</tr>
<tr>
<td>File</td>
<td>VARCHAR(255)</td>
<td>Returns the file name of the database root file, including path.</td>
</tr>
<tr>
<td>ConnCount</td>
<td>INTEGER</td>
<td>Returns the number of connections to the database. The property value does</td>
</tr>
<tr>
<td></td>
<td></td>
<td>not include connections used for internal operations, but it does include</td>
</tr>
<tr>
<td></td>
<td></td>
<td>connections used for events and external environment support.</td>
</tr>
<tr>
<td>PageSize</td>
<td>INTEGER</td>
<td>Returns the page size of the database, in bytes.</td>
</tr>
<tr>
<td>LogName</td>
<td>VARCHAR(255)</td>
<td>Returns the file name of the transaction log, including path.</td>
</tr>
</tbody>
</table>

**Remarks**

If you specify a database ID, sa_db_info returns a single row containing the Number, Alias, File, ConnCount, PageSize, and LogName for the specified database.

If `dbidparm` is greater than zero, then properties for the supplied database are returned. If `dbidparm` is less than zero, then properties for the current database are returned. If `dbidparm` is not supplied or is NULL, then properties for all databases running on the database server are returned.

**Cloud note:** Because of tenant database isolation rules, when this system procedure is run in the cloud it returns only information about the current tenant database.

**Privileges**

No privileges are required to execute this system procedure for the current database. To execute this system procedure for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.
Side effects

None

See also

- “sa_db_properties system procedure” on page 1121
- “List of database properties” [SQL Anywhere Server - Database Administration]

Example

The following statement returns a row for each database that is running on the server:

```
CALL sa_db_info( );
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>Alias</td>
<td>demo</td>
</tr>
<tr>
<td>File</td>
<td>C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.db;</td>
</tr>
<tr>
<td>ConnCount</td>
<td>3</td>
</tr>
<tr>
<td>PageSize</td>
<td>4096</td>
</tr>
<tr>
<td>LogName</td>
<td>C:\Users\Public\Documents\SQL Anywhere 16\Samples\demo.log</td>
</tr>
</tbody>
</table>

**sa_db_list system procedure**

Returns a database ID.

Syntax

```
sa_db_list( [ dbidparm ] )
```

Arguments

- `dbidparm` Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>INTEGER</td>
<td>The database ID number.</td>
</tr>
</tbody>
</table>
Remarks

If `dbidparm` is greater than zero, then the ID for the supplied database is returned. If `dbidparm` is less than zero, then the ID for the current database is returned. If `dbidparm` is not supplied or is NULL, then IDs for all databases running on the database server are returned.

* Cloud note: Because of tenant database isolation rules, when this system procedure is run in the cloud it returns only information about the current tenant database.

Privileges

No privileges are required to execute this system procedure for the current database. To execute this system procedure for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

Side effects

None

See also

- “sa_conn_list system procedure” on page 1109
- “sa_conn_options system procedure” on page 1110
- “DB_NAME function [System]” on page 217

Example

The following example returns the number of databases running on a database server.

```
SELECT count(*) FROM sa_db_list();
```

The following example uses the system procedure `sa_db_list` and the function `DB_Name` to return a list of the databases running on a database server.

```
SELECT DB_NAME(Number) FROM sa_db_list();
```

**sa_db_option system procedure**

Overrides a database option while the database is running.

Syntax

```
sa_db_option(
    opt, val
)
```

Arguments

- **opt** Use this CHAR(128) parameter to specify a database option name.
- **val** Use this CHAR(128) parameter to specify the new value for the database option.
Remarks

Database administrators can use this procedure to override some database options temporarily, without restarting the database.

The option values that are changed using this procedure are reset to their default values when the database shuts down. To change an option value every time the database is started, specify the corresponding database option when the database is started (if one exists).

The following option settings can be changed:

<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Default</th>
<th>System privilege</th>
<th>Database option</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiskSandbox</td>
<td>ON, OFF</td>
<td>OFF</td>
<td>SERVER OPERATOR</td>
<td>“-sbx database option” [SQL Anywhere Server - Database Administration]</td>
</tr>
</tbody>
</table>

- **DiskSandbox** When set to ON, restricts read-write file operations on the database to the directory where the main database file is located and any subdirectories of this directory. To use the sa_db_option system procedure to change disk sandbox settings, you must provide the secure feature key for the manage_disk_sandbox secure feature.

Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

None.

See also

- “-sbx database server option” [SQL Anywhere Server - Database Administration]
- “disk_sandbox option” [SQL Anywhere Server - Database Administration]
- “sa_server_option system procedure” on page 1222

Example

In order for the following example to work, the server must be started with the option: -sk securefkey.

This example enables the SYSTEM secure feature set which includes MANAGE_KEYS, creates a new secure feature key called SECURITY with case-sensitive authorization key newsecuritykey, and then uses the new secure feature key to enable the DiskSandbox option.

```
CALL sp_use_secure_feature_key( 'system', 'securefkey' );
CALL sp_create_secure_feature_key( 'security', 'newsecuritykey', 'manage_security' );
CALL sp_use_secure_feature_key( 'security', 'newsecuritykey' );
CALL sa_db_option( 'DiskSandbox', 'on' );
```

**sa_db_properties system procedure**

Reports database property information.
Syntax

    sa_db_properties( [ dbidparm ] )

Arguments

- **dbidparm**  Use this optional INTEGER parameter to specify the database ID number. The default is NULL.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>INTEGER</td>
<td>The database ID number.</td>
</tr>
<tr>
<td>PropNum</td>
<td>INTEGER</td>
<td>The database property number.</td>
</tr>
<tr>
<td>PropName</td>
<td>VARCHAR(255)</td>
<td>The database property name.</td>
</tr>
<tr>
<td>PropDescription</td>
<td>VARCHAR(255)</td>
<td>The database property description.</td>
</tr>
<tr>
<td>Value</td>
<td>LONG VARCHAR</td>
<td>The database property value.</td>
</tr>
</tbody>
</table>

Remarks

If you specify a database ID, the sa_db_properties system procedure returns the database ID number and the PropNum, PropName, PropDescription, and Value for each available database property. Values are returned for all database properties and statistics related to databases. Valid properties with NULL values are also returned.

If \textit{dbidparm} is greater than zero, then database properties for the supplied database are returned. If \textit{dbidparm} is less than zero, then database properties for the current database are returned. If \textit{dbidparm} is not supplied or is NULL, then database properties for all databases running on the database server are returned.

Privileges

No privileges are required to execute this system procedure for the current database. To execute this system procedure for other databases, you must have either the SERVER OPERATOR or MONITOR system privilege.

Side effects

None

See also

- “sa_db_info system procedure” on page 1118
- “List of database properties” [SQL Anywhere Server - Database Administration]
Example
The following example uses the `sa_db_properties` system procedure to return a result set summarizing database properties for all databases when the invoker has SERVER OPERATOR or MONITOR system privilege. Otherwise, database properties for the current database are returned.

```sql
CALL sa_db_properties();
```

| Number | PropNum | PropName | ...
|--------|---------|----------|--------
| 0      | 0       | ConnCount | ...
| 0      | 1       | IdleCheck | ...
| 0      | 2       | IdleWrite | ...
| ...    | ...     | ...      | ...

The following example uses the `sa_db_properties` system procedure to return a result set summarizing database properties for a second database.

```sql
CALL sa_db_properties(1);
```

### `sa_dependent_views` system procedure

Returns the list of all dependent views for a given table or view.

**Syntax**

```sql
sa_dependent_views(
    [ tbl_name ]
    [, owner_name ]
)
```

**Arguments**

- **`tbl_name`** Use this optional CHAR(128) parameter to specify the name of the table or view. The default is NULL.

- **`owner_name`** Use this optional CHAR(128) parameter to specify the owner for `tbl_name`. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INTEGER</td>
<td>The object ID of the table or view.</td>
</tr>
<tr>
<td>dep_view_id</td>
<td>UNSIGNED INTEGER</td>
<td>The object ID of the dependent views.</td>
</tr>
</tbody>
</table>
Remarks
Use this procedure to obtain the list of IDs of tables and their dependent views.

No errors are generated if no existing tables satisfy the specified criteria for table and owner names. The following conditions also apply:

- If both owner and tbl_name are NULL, information is returned on all tables that have dependent views.
- If tbl_name is NULL but owner is specified, information is returned on all tables owned by the specified owner.
- If tbl_name is specified but owner is NULL, information is returned on any one of the tables with the specified name.

Privileges
None

Side effects
None

See also
- “SYSDEPENDENCY system view” on page 1355
- “View dependencies” [SQL Anywhere Server - SQL Usage]

Examples
In this example, the sa_dependent_views system procedure is used to obtain the list of IDs for the views that are dependent on the SalesOrders table. The procedure returns the table_id for SalesOrders, and the dep_view_id for the dependent view, ViewSalesOrders.

```
CALL sa_dependent_views( 'SalesOrders' );
```

In this example, the sa_dependent_views system procedure is used in a SELECT statement to obtain the list of names of views dependent on the SalesOrders table. The procedure returns the ViewSalesOrders view.

```
SELECT t.table_name FROM SYSTAB t,
sa_dependent_views( 'SalesOrders' ) v
WHERE t.table_id = v.dep_view_id;
```

**sa_describe_cursor system procedure**

Describes the name and type information for the columns of a cursor.

Syntax
```
sa_describe_cursor( cursor_name )
```
Arguments

- **cursor_name**  This VARCHAR(256) value identifies the open cursor to describe.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_number</td>
<td>INTEGER</td>
<td>The ordinal position of the column described by this row, starting at 1.</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>domain_id</td>
<td>SMALLINT</td>
<td>The data type of the column.</td>
</tr>
<tr>
<td>domain_name</td>
<td>VARCHAR(128)</td>
<td>The data type name of the column.</td>
</tr>
<tr>
<td>domain_name_with_size</td>
<td>VARCHAR(160)</td>
<td>The data type name, including size and precision (as used in CREATE TABLE or CAST functions).</td>
</tr>
<tr>
<td>width</td>
<td>INTEGER</td>
<td>The length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.</td>
</tr>
<tr>
<td>scale</td>
<td>INTEGER</td>
<td>The number of digits after the decimal point for numeric data type columns, and zero for all other data types.</td>
</tr>
<tr>
<td>declared_width</td>
<td>INTEGER</td>
<td>The length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.</td>
</tr>
<tr>
<td>user_type_id</td>
<td>SMALLINT</td>
<td>The user-defined data type if applicable, otherwise NULL.</td>
</tr>
<tr>
<td>user_type_name</td>
<td>VARCHAR(128)</td>
<td>The user-defined data type if applicable, otherwise NULL.</td>
</tr>
<tr>
<td>correlation_name</td>
<td>VARCHAR(128)</td>
<td>The correlation name associated with the expression if applicable, otherwise NULL.</td>
</tr>
<tr>
<td>base_table_id</td>
<td>UNSIGNED INTEGER</td>
<td>The table_id if the expression is a column, otherwise NULL.</td>
</tr>
<tr>
<td>base_column_id</td>
<td>UNSIGNED INTEGER</td>
<td>The column_id if the expression is a column, otherwise NULL.</td>
</tr>
<tr>
<td>base_owner_name</td>
<td>VARCHAR(128)</td>
<td>The owner name if the expression is a column, otherwise NULL.</td>
</tr>
</tbody>
</table>
### Remarks

The sa_describe_cursor system procedure provides an API-independent mechanism for retrieving the description of the columns returned by the cursor. The system procedure can be useful when writing stored procedures that work with dynamic SQL.

The sa_describe_cursor system procedure can be used in a CALL statement or in the FROM clause of a SELECT statement.

`cursor_name` must refer to an open cursor in the current connection. Use the sa_list_cursors system procedure to get the list of open cursors for the connection.

### Privileges

None

### Side effects

None

### See also

- “sa_list_cursors system procedure” on page 1166
- “sa_copy_cursor_to_temp_table system procedure” on page 1115
- “SYSDOMAIN system view” on page 1356
- “SYSUSER system view” on page 1412
- “SYSTABCOL system view” on page 1401
- “SYSUSERTYPE system view” on page 1415
- “SYSDOMAIN system view” on page 1356
Example
The following batch creates a cursor named myCursor on the Products table and then opens it. The sa_describe_cursor system procedure is used to describe the columns of the cursor. A result set containing 5 rows, one for each column, is produced.

```
BEGIN
  DECLARE myCursor CURSOR FOR
  SELECT ID, Name, Description, Color, Quantity FROM Products;
  OPEN myCursor;
  CALL sa_describe_cursor( 'myCursor' );
  CLOSE myCursor;
END
```

**sa_describe_query system procedure**
Describes the result set for a query with one row describing each output column of the query.

**Syntax**

```
sa_describe_query(
  query
  [, add_keys ]
)
```

**Arguments**

- **query** Use this LONG VARCHAR parameter to specify the text of the SQL statement being described.

- **add_keys** Use this optional BIT parameter to specify whether to determine a set of columns that uniquely identify rows in the result set for the query being described. The default is 0; the database server does not attempt to identify the columns. See the Remarks section below for a full explanation of this parameter.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_number</td>
<td>INTEGER</td>
<td>The ordinal position of the column described by this row, starting at 1.</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>domain_id</td>
<td>SMALLINT</td>
<td>The data type of the column.</td>
</tr>
<tr>
<td>domain_name</td>
<td>VARCHAR(128)</td>
<td>The data type name.</td>
</tr>
<tr>
<td>domain_name_with_size</td>
<td>VARCHAR(160)</td>
<td>The data type name, including size and precision (as used in CREATE TABLE or CAST functions).</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>width</td>
<td>INTEGER</td>
<td>The length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.</td>
</tr>
<tr>
<td>scale</td>
<td>INTEGER</td>
<td>The number of digits after the decimal point for numeric data type columns, and zero for all other data types.</td>
</tr>
<tr>
<td>declared_width</td>
<td>INTEGER</td>
<td>The length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.</td>
</tr>
<tr>
<td>user_type_id</td>
<td>SMALLINT</td>
<td>The type_id of the user-defined data type if there is one, otherwise NULL.</td>
</tr>
<tr>
<td>user_type_name</td>
<td>VARCHAR(128)</td>
<td>The name of the user-defined data type if there is one, otherwise NULL.</td>
</tr>
<tr>
<td>correlation_name</td>
<td>VARCHAR(128)</td>
<td>The correlation name associated with the expression if one is available, otherwise NULL.</td>
</tr>
<tr>
<td>base_table_id</td>
<td>UNSIGNED INTEGER</td>
<td>The table_id if the expression is a column, otherwise NULL.</td>
</tr>
<tr>
<td>base_column_id</td>
<td>UNSIGNED INTEGER</td>
<td>The column_id if the expression is a column, otherwise NULL.</td>
</tr>
<tr>
<td>base_owner_name</td>
<td>VARCHAR(128)</td>
<td>The owner name if the expression is a column, otherwise NULL.</td>
</tr>
<tr>
<td>base_table_name</td>
<td>VARCHAR(128)</td>
<td>The table name if the expression is a column, otherwise NULL.</td>
</tr>
<tr>
<td>base_column_name</td>
<td>VARCHAR(128)</td>
<td>The column name if the expression is a column, otherwise NULL.</td>
</tr>
<tr>
<td>nulls_allowed</td>
<td>BIT</td>
<td>An indicator that is 1 if the expression can be NULL, otherwise 0.</td>
</tr>
<tr>
<td>is_autoincrement</td>
<td>BIT</td>
<td>An indicator that is 1 if the expression is a column declared to be AUTOINCREMENT, otherwise 0.</td>
</tr>
<tr>
<td>is_key_column</td>
<td>BIT</td>
<td>An indicator that is 1 if the expression is part of a key for the result set, otherwise 0. For more information, see the Remarks section below.</td>
</tr>
</tbody>
</table>
Remarks

The sa_describe_query procedure provides an API-independent mechanism to describe the name and type information for the expressions in the result set of a query.

When 1 is specified for add_keys, the sa_describe_query procedure attempts to find a set of columns from the objects being queried that, when combined, can be used as a key to uniquely identify rows in result set of the query being described. The key takes the form of one or more columns from the objects being queried, and may include columns that are not explicitly referenced in the query. If the optimizer finds a key, the column or columns used in the key are identified in the results by an is_key_column value of 1. If no key is found, an error is returned.

For any column that is included in the key but that is not explicitly referenced in the query, the is_added_key_column value is set to 1 to indicate that the column has been added to the results for the procedure; otherwise, the value of is_added_key_column is 0.

If you do not specify add_keys, or you specify a value of 0, the optimizer does not attempt to find a key for the result set, and the is_key_column and is_added_key_column columns contain NULL.

The declared_width and width values both describe the size of a column. The declared_width describes the size of the column as defined by the CREATE TABLE statement or by the query, while the width value gives the size of the column when fetched to the client. The client representation of a type may be different from the database server. For example, date and time types are converted to strings if the return_date_time_as_string option is on. For strings, columns declared with character-length semantics have a declared_width value that matches the CREATE TABLE size, while the width value gives the maximum number of bytes needed to store the returned string. For example:

<table>
<thead>
<tr>
<th>Declaration</th>
<th>width</th>
<th>declared_width</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(10)</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>CHAR(10 CHAR)</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>depends on the length of the timestamp format string</td>
<td>8</td>
</tr>
<tr>
<td>NUMERIC(10, 3)</td>
<td>10 (precision)</td>
<td>10 (precision)</td>
</tr>
</tbody>
</table>

Privileges

None

Side effects

None
See also

- “EXPRTYPE function [Miscellaneous]” on page 246
- “Character data types” on page 89
- “return_date_time_as_string option” [SQL Anywhere Server - Database Administration]
- “SYSDOMAIN system view” on page 1356
- “SYSDOMAIN system view” on page 1356
- “SYSUSERTYPE system view” on page 1415
- “SYSTAB system view” on page 1399
- “SYSUSER system view” on page 1412

Examples

The following example describes the information returned when querying all columns in the Departments table:

```
SELECT *
FROM sa_describe_query( 'SELECT * FROM Departments DEPT' );
```

The results show the values of the is_key_column and is_added_key_column as NULL because the add_keys parameter was not specified.

The following example describes the information returned by querying the DepartmentName and Surname columns of the Employees table, joined with the Departments table:

```
SELECT *
FROM sa_describe_query( 'SELECT DepartmentName, Surname
FROM Employees E JOIN Departments D ON E.EmployeeID = D.DepartmentHeadId',
add_keys = 1 );
```

The results shows a 1 in rows 3 and 4 of the result set, indicating that the columns needed to uniquely identify rows in the result set for the query are Employees.EmployeeID and Departments.DepartmentID. Also, a 1 is present in the is_added_key_column for rows 3 and 4 because Employees.EmployeeID and Departments.DepartmentID were not explicitly referenced in the query being described.

**sa_describe_shapefile system procedure**

Describes the names and types of columns contained in an ESRI shapefile. This system feature is for use with the spatial data features.

**Syntax**

```
sa_describe_shapefile(
    shp_filename
    , srid
    [, encoding ]
)
```

**Arguments**

- **shp_filename** A VARCHAR(512) parameter that identifies the location of the ESRI shapefile. The file name must have the extension .shp and must have an associated .dbf file with the same base name located in the same directory. The path is relative to the database server, not the client application.

1130Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
- **srid**  An INTEGER parameter that identifies the SRID for the geometries in the shapefile. Specify NULL to indicate the column can store multiple SRIDs. Specifying NULL limits the operations that can be performed on the geometry values.

- **encoding**  An optional VARCHAR(50) parameter that identifies the encoding to use when reading the shapefile. The default is NULL. When encoding is NULL, the ISO-8859-1 character set is used.

### Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_number</td>
<td>INTEGER</td>
<td>The ordinal position of the column described by this row, starting at 1.</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>domain_name_with_size</td>
<td>VARCHAR(160)</td>
<td>The data type name, including size and precision (as used in CREATE TABLE or CAST functions).</td>
</tr>
</tbody>
</table>

### Remarks

The sa_describe_shapefile system procedure is used to describe the name and type of columns in an ESRI shapefile. This information can be used to create a table to load data from a shapefile using the LOAD TABLE or INPUT statements. Alternately, this system procedure can be used to read a shapefile by specifying the WITH clause for OPENSTRING...FORMAT SHAPEFILE.

### Privileges

- If the -gl database option is set to DBA, you must have one of the following system privileges:
  - ALTER ANY TABLE
  - ALTER ANY OBJECT
  - LOAD ANY TABLE
  - READ FILE

- If the -gl database option is set to ALL, no privileges are required.

- If the -gl database option is set to NONE, you must have the READ FILE system privilege.

### See also

- “LOAD TABLE statement” on page 873
- “INPUT statement [Interactive SQL]” on page 854
- “Support for ESRI shapefiles” [SQL Anywhere Server - Spatial Data Support]

### Example

The following example displays a string that was used to create a table for storing shapefile data:

```sql
BEGIN
    DECLARE create_cmd LONG VARCHAR;
    SELECT 'create table if not exists esri_load( record_number int primary key, ' ||
```
(SELECT list( name || ' ' || domain_name_with_size, ', ' ORDER BY
column_number )
FROM sa_describe_shapefile( 'c:\esri\tgr36069trt00.shp', 1000004326 )
WHERE column_number > 1 || ' ')
INTO create_cmd;
EXECUTE IMMEDIATE create_cmd;
END

You can load the shapefile data into the table using the following statement (provided that you have the
LOAD ANY TABLE system privilege and that the -gl database option has not been set to NONE):

LOAD TABLE esri_load
USING FILE 'c:\esri\tgr36069trt00.shp'
FORMAT SHAPEFILE;

sa_disable_auditing_type system procedure
Disables auditing of specific events.

Syntax
sa_disable_auditing_type( types )

Arguments
- **types** Use this VARCHAR(128) parameter to specify a comma-delimited string containing one or
more of the following values:
  - all disables all types of auditing.
  - connect disables auditing of both successful and failed connection attempts.
  - connectFailed disables auditing of failed connection attempts.
  - DDL disables auditing of DDL statements.
  - options disables auditing of public options.
  - permission disables auditing of permission checks, user checks, and SETUSER statements.
  - permissionDenied disables auditing of failed permission and user checks.
  - triggers disables auditing in response to trigger events.

Remarks
You can use the sa_disable_auditing_type system procedure to disable auditing of one or more categories
of information.

Setting this option to all disables all auditing. You can also disable auditing by setting the
PUBLIC.auditing option to Off.
Privileges

You must have the SET ANY SECURITY OPTION system privilege.

Side effects

None

See also

- “sa_enable_auditing_type system procedure” on page 1134
- “Database activity audits” [SQL Anywhere Server - Database Administration]
- “auditing option” [SQL Anywhere Server - Database Administration]

Example

To disable all auditing:

```sql
CALL sa_disable_auditing_type( 'all' );
```

**sa_disk_free_space system procedure**

Reports information about space available for a dbspace, transaction log, transaction log mirror, and/or temporary file.

**Syntax**

```sql
sa_disk_free_space( p_dbspace_name )
```

**Arguments**

- `p_dbspace_name`
  
  Use this VARCHAR(128) parameter to specify the name of a dbspace, transaction log file, transaction log mirror file, or temporary file. The default is NULL.

  If there is a dbspace called log, mirror, or temp, you can prefix the keyword with an underscore. For example, use _log to get information about the log file if a dbspace called log exists.

  Specify SYSTEM to get information about the main database file, TEMPORARY or TEMP to get information about the temporary file, TRANSLOG to get information about the transaction log, or TRANSLOGMIRROR to get information about the transaction log mirror.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbspace_name</td>
<td>VARCHAR(128)</td>
<td>This is the dbspace name, transaction log file, transaction log mirror file, or temporary file.</td>
</tr>
<tr>
<td>free_space</td>
<td>UNSIGNED BIGINT</td>
<td>The number of free bytes on the volume.</td>
</tr>
<tr>
<td>total_space</td>
<td>UNSIGNED BIGINT</td>
<td>The total amount of disk space available on the drive where the dbspace resides.</td>
</tr>
</tbody>
</table>
Remarks
If the \texttt{p\_dbspace\_name} parameter is not specified or is NULL, then the result set contains one row for
each dbspace, plus one row for each of the transaction log, transaction log mirror, and temporary file, if
they exist. If \texttt{p\_dbspace\_name} is specified, then exactly one or zero rows are returned (zero if no such
dbspace exists, or if log or mirror is specified and there is no log or mirror file).

Privileges
You must have the MANAGE ANY DBSPACE system privilege.

Side effects
None

Example
The following example uses the \texttt{sa\_disk\_free\_space} system procedure to return a result set containing
information about available space.

\begin{verbatim}
CALL sa_disk_free_space();
\end{verbatim}

\begin{center}
\begin{tabular}{|l|l|l|}
\hline
\texttt{dbspace\_name} & \texttt{free\_space} & \texttt{total\_space} \\
\hline
system & 10952101888 & 21410402304 \\
translog & 10952101888 & 21410402304 \\
temporary & 10952101888 & 21410402304 \\
\hline
\end{tabular}
\end{center}

See also
\begin{itemize}
\item “Predefined dbspaces” [SQL Anywhere Server - Database Administration]
\end{itemize}

\textbf{sa\_enable\_auditing\_type} system procedure
Enables auditing and specifies which events to audit.

Syntax
\texttt{sa_enable_auditing_type( types )}

Arguments
\begin{itemize}
\item \texttt{types} Use this VARCHAR(128) parameter to specify a comma-delimited string containing one or
more of the following values:
  \begin{itemize}
  \item \texttt{all} enables all types of auditing.
  \item \texttt{connect} enables auditing of both successful and failed connection attempts.
  \item \texttt{connectFailed} enables auditing of failed connection attempts.
  \end{itemize}
\end{itemize}
○ **DDL** enables auditing of DDL statements.

○ **options** enables auditing of public options.

○ **permission** enables auditing of permission checks, user checks, and SETUSER statements.

○ **permissionDenied** enables auditing of failed permission and user checks.

○ **triggers** enables auditing after a trigger event.

**Remarks**

sa_enable_auditing_type works with the PUBLIC.auditing option to enable auditing of specific types of information.

If you set the PUBLIC.auditing option to On, and do not specify which type of information to audit, the default setting (all) takes effect. In this case, all types of auditing information are recorded.

If you set the PUBLIC.auditing option to On, and disable all types of auditing using sa_disable_auditing_type, no auditing information is recorded. To re-establish auditing, you must use sa_enable_auditing_type to specify which type of information you want to audit.

If you set the PUBLIC.auditing option to Off, then no auditing information is recorded, regardless of the sa_enable_auditing_type setting.

**Privileges**

You must have the SET ANY SECURITY OPTION system privilege.

**Side effects**

None

**See also**

- “sa_disable_auditing_type system procedure” on page 1132
- “Database activity audits” [SQL Anywhere Server - Database Administration]
- “auditing option” [SQL Anywhere Server - Database Administration]

**Example**

To enable only option auditing:

```
CALL sa_enable_auditing_type( 'options' );
```

**sa_eng_properties system procedure**

Reports database server property information.

**Syntax**

```
sa_eng_properties( )
```
Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropNum</td>
<td>INTEGER</td>
<td>The database server property number.</td>
</tr>
<tr>
<td>PropName</td>
<td>VARCHAR(255)</td>
<td>The database server property name.</td>
</tr>
<tr>
<td>PropDescription</td>
<td>VARCHAR(255)</td>
<td>The database server property description.</td>
</tr>
<tr>
<td>Value</td>
<td>LONG VARCHAR</td>
<td>The database server property value.</td>
</tr>
</tbody>
</table>

Remarks

Returns the PropNum, PropName, PropDescription, and Value for each available server property. Values are returned for all database server properties and statistics related to database servers.

Privileges

None

Side effects

None

See also

- “List of database server properties” [SQL Anywhere Server - Database Administration]
- “System functions” on page 154

Example

The following statement returns a set of available server properties

```sql
CALL sa_eng_properties( );
```

| PropNum | PropName | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IdleWrite</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>IdleChkPt</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**sa_error_stack_trace system procedure**

Returns the stack trace of the error that invoked the error handler.

Syntax

```sql
sa_error_stack_trace( )
```
## Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackLevel</td>
<td>UNSIGNED SMALLINT</td>
<td>The line number of the stack (1 for the top line). The statement that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>generated the error has the highest number.</td>
</tr>
<tr>
<td>UserName</td>
<td>CHAR(128)</td>
<td>The name of the owner of the procedure or NULL if the current level is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in a batch.</td>
</tr>
<tr>
<td>ProcName</td>
<td>CHAR(128)</td>
<td>The name of the procedure where the statement was executed, or the batch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type.</td>
</tr>
<tr>
<td>LineNumber</td>
<td>UNSIGNED INTEGER</td>
<td>The line number of the call within the procedure.</td>
</tr>
<tr>
<td>IsResignal</td>
<td>BIT</td>
<td>1 if the statement is a resignal, and 0 otherwise.</td>
</tr>
</tbody>
</table>

## Remarks

Each row in the result set represents a single call on the call stack of the error. If the compound statement is not part of a procedure, function, trigger, or event, the type of batch (watcom_batch or tsql_batch) is returned instead of the procedure name.

This procedure returns the same information as the ERROR_STACK_TRACE function.

## Privileges

None

## Side effects

None.

## See also

- “TRY statement” on page 1022
- “BEGIN statement” on page 523
- “ERROR_LINE function [Miscellaneous]” on page 228
- “ERROR_MESSAGE function [Miscellaneous]” on page 229
- “ERROR_PROCEDURE function [function type]” on page 230
- “ERROR_SQLCODE function [Miscellaneous]” on page 231
- “ERROR_SQLSTATE function [Miscellaneous]” on page 232
- “ERROR_STACK_TRACE function [Miscellaneous]” on page 233
- “STACK_TRACE function [Miscellaneous]” on page 371
- “sa_stack_trace system procedure” on page 1250
- “Nested compound statements and exception handlers” [SQL Anywhere Server - SQL Usage]

## Example

This example shows an example of calling the sa_error_stack_trace system procedure:

```sql
CALL sa_error_stack_trace();
```
This example shows the output of the sa_error_stack_trace system procedure with RESIGNAL:

```sql
CREATE OR REPLACE PROCEDURE error_reporting_procedure()
BEGIN
    SELECT *
    FROM sa_error_stack_trace();
END;

CREATE OR REPLACE PROCEDURE proc1()
BEGIN TRY
    BEGIN TRY
        DECLARE v INTEGER = 0;
        SET v = 1 / v;
    END TRY
    BEGIN CATCH
        CALL proc2();
    END CATCH
END TRY
BEGIN CATCH
    CALL error_reporting_procedure();
END CATCH;

CREATE OR REPLACE PROCEDURE proc2()
BEGIN
    CALL proc3();
END;

CREATE OR REPLACE PROCEDURE proc3()
BEGIN
    RESIGNAL;
END;

CALL proc1();
```

When the example above is run, the following result set is produced:

<table>
<thead>
<tr>
<th>StackLevel</th>
<th>UserName</th>
<th>ProcName</th>
<th>LineNumber</th>
<th>IsResignal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DBA</td>
<td>proc1</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>DBA</td>
<td>proc2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>DBA</td>
<td>proc3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>DBA</td>
<td>proc1</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

**sa_event_schedules system procedure**

Displays schedule information about events.

**Syntax**

```sql
sa_event_schedules( evt_id )
```

**Arguments**

- **evt_id**  
  This INTEGER parameter takes the ID number of an event.
Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sched_name</td>
<td>VARCHAR(128)</td>
<td>The name of the schedule the event runs under.</td>
</tr>
<tr>
<td>sched_def</td>
<td>LONG VARCHAR</td>
<td>The schedule definition.</td>
</tr>
</tbody>
</table>

Remarks

This procedure returns information about a specified event, including when the event is scheduled to run and how often it runs. If the event does not have a schedule, the procedure does not return a result set.

Privileges

None

Side effects

None

See also

- “CREATE EVENT statement” on page 570
- “EVENT_CONDITION function [System]” on page 238
- “EVENT_CONDITION_NAME function [System]” on page 240
- “EVENT_PARAMETER function [System]” on page 240

Example

The following example retrieves the event schedule for the IncrementalBackup event.

```
call sa_event_schedules( (SELECT event_id FROM SYSEVENT WHERE event_name='IncrementalBackup') );
```

<table>
<thead>
<tr>
<th>sched_name</th>
<th>sched_def</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncrementalBackup</td>
<td>START TIME '01:00:00' EVERY 24 HOURS</td>
</tr>
</tbody>
</table>

sa_external_library_unload system procedure

Unloads an external library.

Syntax

```
sa_external_library_unload( [ lib_name ] )
```

Arguments

- **lib_name**  Use this optional LONG VARCHAR parameter to specify the name of a library to be unloaded. If no library is specified, all external libraries that are not in use are unloaded. The default is NULL.
Remarks
If an external library is specified, but is in use or is not loaded, an error is returned. If no parameter is specified, no error is returned.

The library name must match exactly the path and letter case of the original library specification. For example, the library name must match exactly the string following the '@' in the following EXTERNAL NAME clause.

```
EXTERNAL NAME 'xp_replicate@c:\sqlany\samples\sqlanywhere\ExternalProcedures\extproc.dll'
```

Privileges
You must have the MANAGE ANY EXTERNAL OBJECT system privilege.

Side effects
None

See also
● “SQL Anywhere external call interface” [SQL Anywhere Server - Programming]

Example
The following example unloads an external library called extproc.dll:
```
CALL sa_external_library_unload( 'extproc.dll' );
```

The following example unloads all libraries that are not currently in use:
```
CALL sa_external_library_unload();
```

sa_flush_cache system procedure
Empties all pages for the current database in the database server cache.

Syntax
```
sa_flush_cache( )
```

Remarks
Database administrators can use this procedure to empty the contents of the database server cache for the current database. This is useful in performance measurement to ensure repeatable results.

Privileges
You must have the SERVER OPERATOR system privilege.

Side effects
None
Example

The following example empties all pages for the current database in the database server cache.

```sql
CALL sa_flush_cache();
```

**sa_flush_statistics system procedure**

Saves all cost model statistics in the database server cache.

**Syntax**

```sql
sa_flush_statistics()
```

**Remarks**

Use this procedure to flush current cost model statistics in the database, currently cached, to disk. You can then retrieve the statistics using the `sa_get_histogram` system procedure, or the Histogram utility (dbhist). When this system procedure runs, the ISYSCOLSTAT system table is updated. Under normal operation it should not be necessary to execute this procedure because the server automatically writes out statistics to disk on a periodic basis.

**Privileges**

You must have the MANAGE ANY STATISTICS or SERVER OPERATOR system privilege.

**Side effects**

None

**See also**

- “`sa_get_histogram` system procedure” on page 1145
- “SYSCOLSTAT system view” on page 1351
- “Histogram utility (dbhist)” [SQL Anywhere Server - Database Administration]

**Example**

The following example saves all cost model statistics in the database server cache.

```sql
CALL sa_flush_statistics();
```

**sa_get_bits system procedure**

Takes a bit string and returns a row for each bit in the string. By default, only rows with a bit value of 1 are returned.

**Syntax**

```sql
sa_get_bits(
    bit_string
    [, only_on_bits ]
)
```
Arguments
- **bit_string**  Use this LONG VARBIT parameter to specify the bit string from which to get the bits. If the `bit_string` parameter is NULL, no rows are returned.

- **only_on_bits**  Use this optional BIT parameter to specify whether to return only rows with on bits (bits with the value of 1). Specify 1 (the default) to return only rows with on bits; specify 0 to return rows for all bits in the bit string.

Result set

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bitnum</td>
<td>UNSIGNED INTEGER</td>
<td>The position of the bit described by this row. For example, the first bit in the bit string has bitnum of 1.</td>
</tr>
<tr>
<td>bit_val</td>
<td>BIT</td>
<td>The value of the bit at position bitnum. If <code>only_on_bits</code> is set to 1, this value is always 1.</td>
</tr>
</tbody>
</table>

Remarks
The `sa_get_bits` system procedure decodes a bit string, returning one row for each bit in the bit string, indicating the value of the bit. If `only_on_bits` is set to 1 (the default) or NULL, then only rows corresponding to on bits are returned. An optimization allows this case to be processed efficiently for long bit strings that have few on bits. If `only_on_bits` is set to 0, then a row is returned for each bit in the bit string.

For example, the statement `CALL sa_get_bits( '1010' )` returns the following result set, indicating on bits in positions 1 and 3 of the bit string.

<table>
<thead>
<tr>
<th>bitnum</th>
<th>bit_val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

The `sa_get_bits` system procedure can be used to convert a bit string into a relation. This can be used to join a bit string with a table, or to retrieve a bit string as a result set instead of as a single binary value. It can be more efficient to retrieve a bit string as a result set if there are a large number of 0 bits, as these do not need to be retrieved.

Privileges
None

Side effects
None
See also
  ● “sa_split_list system procedure” on page 1248
  ● “SET_BIT function [Bit array]” on page 359
  ● “SET_BITS function [Aggregate]” on page 360
  ● “GET_BIT function [Bit array]” on page 249

Examples
The following example shows how to use the sa_get_bits system procedure to encode a set of integers as a
bit string, and then decode it for use in a join:

```sql
CREATE VARIABLE @s_depts LONG VARBIT;

SELECT  SET_BITS( DepartmentID )
  INTO @s_depts
FROM Departments
WHERE DepartmentName like 'S%';

SELECT *
FROM sa_get_bits( @s_depts ) B
  JOIN Departments D ON B.bitnum = D.DepartmentID;
```

sa_get_dtt system procedure
Reports the current value of the Disk Transfer Time (DTT) model, which is part of the cost model.

Syntax

```
sa_get_dtt( file_id )
```

Arguments

- **file_id**  Use this UNSIGNED SMALLINT parameter to specify the database file ID.

Remarks
You can obtain the file_id from the SYSDBSPACE system view.

This procedure, intended for internal diagnostic purposes, retrieves data from the ISYSOPTSTAT system
table.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BandSize</td>
<td>UNSIGNED INTEGER</td>
<td>Size, in pages, of disk over which random access takes place.</td>
</tr>
<tr>
<td>ReadTime</td>
<td>UNSIGNED INTEGER</td>
<td>Amortized cost, in microseconds, of reading one page.</td>
</tr>
<tr>
<td>WriteTime</td>
<td>UNSIGNED INTEGER</td>
<td>Amortized cost, in microseconds, of writing one page.</td>
</tr>
</tbody>
</table>
Privileges
None

Side effects
None

See also
- “SYSDBSPACE system view” on page 1354
- “SYSOPTSTAT system view” on page 1376
- “sa_get_dtt_groupreads system procedure” on page 1144

Example
The following example reports the current value of the Disk Transfer Time (DTT) model for the system dbspace.

```
CALL sa_get_dtt( (select dbspace_id from SYSDBSPACE where
dbspace_name='system') );
```

sa_get_dtt_groupreads system procedure
Estimates and reports the cost of issuing group reads on the database server.

Syntax
```
sa_get_dtt_groupreads( dbspace_id )
```

Arguments
- **dbspace_id** Use this UNSIGNED SMALLINT parameter to specify the database file ID.

Remarks
You can obtain the `dbspace_id` from the SYSDBSPACE system view. The estimates returned by the sa_get_dtt_groupreads system procedure are part of the cost model, and are used to select group reads of appropriate sizes during operations such as sorting.

This procedure, intended for internal diagnostic purposes, retrieves data from the ISYSOPTSTAT system table. If no entries are recorded in this table, typical values are returned. To tailor estimates for your hardware, execute the following statement:

```
ALTER DATABASE CALIBRATE GROUP READ;
```

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupSize</td>
<td>UNSIGNED INTEGER</td>
<td>Size, in pages, of disk over which random access takes place.</td>
</tr>
<tr>
<td>ReadTime</td>
<td>FLOAT</td>
<td>Amortized cost, in microseconds, of reading one page.</td>
</tr>
</tbody>
</table>
Privileges
None

Side effects
None

See also
- “SYSDBSPACE system view” on page 1354
- “SYSOPTSTAT system view” on page 1376
- “ALTER DATABASE statement” on page 426
- “sa_get_dtt system procedure” on page 1143

Example
The following example reports the cost of issuing group reads for the system dbspace.

```
CALL sa_get_dtt_groupreads( (select dbspace_id from SYSDBSPACE where
dbname_name='system') );
```

### sa_get_histogram system procedure

Retrieves the histogram for a column.

**Syntax**

```sql
sa_get_histogram(
    col_name
    , tbl_name
    [ , owner_name ]
)
```

**Arguments**

- **col_name** Use this CHAR(128) parameter to specify the column for which to retrieve the histogram.
- **tbl_name** Use this CHAR(128) parameter to specify the table in which col_name is found.
- **owner_name** Use this optional CHAR(128) parameter to specify the owner of tbl_name. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StepNumber</td>
<td>SMALLINT</td>
<td>Histogram bucket number. The frequency of the first bucket (Step-Number = 0) indicates the selectivity of NULLs.</td>
</tr>
<tr>
<td>Low</td>
<td>CHAR(128)</td>
<td>Lowest (inclusive) column value in the bucket.</td>
</tr>
</tbody>
</table>
### Remarks

This procedure, intended for internal diagnostic purposes, retrieves column statistics from the database server for the specified columns. While these statistics are permanently stored in the ISYSCOLSTAT system table, they are maintained in memory while the server is running, and written to ISYSCOLSTAT periodically. As such, the statistics returned by the sa_get_histogram system procedure may differ from those obtained by selecting from ISYSCOLSTAT at any given point of time.

You can manually update ISYSCOLSTAT with the latest statistics held in memory using the sa_flush_statistics system procedure, however, this is not recommended in a production environment, and should be reserved for diagnostic purposes.

A singleton bucket is indicated by a Low value in the result set being equal to the corresponding High value.

It is recommended that you use the Histogram utility to view histograms.

To determine the selectivity of a predicate over a string column, use the ESTIMATE or ESTIMATE_SOURCE functions. For string columns, both sa_get_histogram and the Histogram utility retrieve nothing from the ISYSCOLSTAT system table. Attempting to retrieve string data generates an error.

Statistics (including histograms) may not be present for a table or materialized view, for example, if statistics were recently dropped. In this case, the result set for the sa_get_histogram system procedure is empty. To create statistics for a table or materialized view, execute a CREATE STATISTICS statement.

### Privileges

You must have the MANAGE ANY STATISTICS system privilege.

### Side effects

None

### See also

- “Optimizer estimates and statistics” [SQL Anywhere Server - SQL Usage]
- “Histogram utility (dbhist)” [SQL Anywhere Server - Database Administration]
- “SYSCOLSTAT system view” on page 1351
- “CREATE STATISTICS statement” on page 682
- “sa_flush_statistics system procedure” on page 1141
- “ESTIMATE_SOURCE function [Miscellaneous]” on page 237
- “ESTIMATE function [Miscellaneous]” on page 236
- “Histogram utility (dbhist)” [SQL Anywhere Server - Database Administration]
Example

For example, the following statement retrieves the histogram for the ProductID column of the SalesOrderItems table:

```
CALL sa_get_histogram( 'ProductID', 'SalesOrderItems' );
```

**sa_get_ldapserver_status system procedure**

Allows you to determine the current status of LDAP servers.

Syntax

```
sa_get_ldapserver_status( )
```

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldsrv_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique number identifying the LDAP server.</td>
</tr>
<tr>
<td>ldsrv_name</td>
<td>CHAR(128)</td>
<td>The name of the LDAP server.</td>
</tr>
<tr>
<td>ldsrv_state</td>
<td>CHAR(9)</td>
<td>The current state of the LDAP server at last check-point.</td>
</tr>
<tr>
<td>ldsrv_last_state_change</td>
<td>TIMESTAMP</td>
<td>The local time that the state was changed.</td>
</tr>
</tbody>
</table>

Remarks

This procedure returns a result set that shows the current status of LDAP servers.

Privileges

None

Side effects

None

See also

- “LDAP user authentication” [SQL Anywhere Server - SQL Usage]

Example

The following example uses the sa_get_ldapserver_status system procedure to return the status of all LDAP servers.

```
CALL sa_get_ldapserver_status;
```
sa_get_request_profile system procedure

Analyzes the request log to determine the execution times of similar statements.

Syntax

```
sa_get_request_profile(
    [ filename
    [, conn_id
    [, first_file
)
```

Arguments

- **filename** Use this optional LONG VARCHAR parameter to specify the request logging file name. The default is NULL.
- **conn_id** Use this optional UNSIGNED INTEGER parameter to specify the connection ID number. The default is 0.
- **first_file** Use this optional INTEGER parameter to specify the first request log file to analyze. The default is -1.
- **num_files** Use this optional INTEGER parameter to specify the number of request log files to analyze. The default is 1.

Remarks

This procedure calls sa_get_request_times to process a request log file, and then summarizes the results into the global temporary table satmp_request_profile. This table contains the statements from the log along with how many times each was executed, and their total, average, and maximum execution times. The table can be sorted in various ways to identify targets for performance optimization efforts.

If you do not specify a log file (filename), the default is the current log file that is specified with the -zo database server option, or that has been specified by

```
    sa_server_option( 'RequestLogFile', filename )
```

If a connection ID is specified, it is used to filter information from the log so that only requests for that connection are retrieved.

Privileges

You must have the DIAGNOSTICS system role, and the MANAGE PROFILING system privilege.

Side effects

Automatic commit
See also

- “sa_get_request_times system procedure” on page 1149
- “sa_statement_text system procedure” on page 1252
- “sa_server_option system procedure” on page 1222
- “-zo database server option” [SQL Anywhere Server - Database Administration]

Example

The following command obtains the request times for the requests in the file req.out.

```sql
CALL sa_get_request_profile('req.out');
```

The following command obtains the request times for the requests in the files req.out.3, req.out.4, and req.out.5.

```sql
CALL sa_get_request_profile('req.out',0,3,3);
```

**sa_get_request_times system procedure**

Analyzes the request log to determine statement execution times.

**Syntax**

```
sa_get_request_times( [ filename
, conn_id
, first_file
, num_files ] )
```

**Arguments**

- **filename** Use this optional LONG VARCHAR parameter to specify the request logging file name. The default is NULL.
- **conn_id** Use this optional UNSIGNED INTEGER parameter to specify the connection ID number. The default is 0.
- **first_file** Use this optional INTEGER parameter to specify the first file to analyze. The default is -1.
- **num_files** Use this optional INTEGER parameter to specify the number of request log files to analyze. The default is 1.

**Remarks**

This procedure reads the specified request log and populates the global temporary table satmp_request_time with the statements from the log and their execution times.

For statements such as inserts and updates, the execution time is straightforward. For queries, the time is calculated from preparing the statement to dropping it, including describing it, opening a cursor, fetching rows, and closing the cursor. For most queries, this is an accurate reflection of the amount of time taken. When the cursor is left open while other events take place, such as operator interaction or client processing, the time appears as a large value but is not a true indication that the query is costly.
This procedure recognizes host variables in the request log and populates the global temporary table satmp_request_hostvar with their values. For older databases where this temporary table does not exist, host variable values are ignored.

If you do not specify a log file, the default is the current log file that is specified in the command with -zo, or that has been specified by:

```sql
call sa_server_option( 'RequestLogFile', filename )
```

If a connection ID is specified, it is used to filter information from the log so that only requests for that connection are retrieved.

**Privileges**

You must have the MANAGE PROFILING or MONITOR system privilege.

**Side effects**

Automatic commit

**Example**

The following command obtains the execution times for the requests in the file `req.out`.

```sql
CALL sa_get_request_times('req.out');
```

The following command obtains the execution times for the requests in the files `req.out.3`, `req.out.4`, and `req.out.5`.

```sql
CALL sa_get_request_times('req.out',0,3,3);
```

**See also**

- “sa_get_request_profile system procedure” on page 1148
- “sa_statement_text system procedure” on page 1252
- “sa_server_option system procedure” on page 1222

**sa_get_server_messages system procedure [deprecated]**

Allows you to return constants from the database server messages window as a result set.

This system procedure is deprecated. Use `sa_server_messages` instead.

**Syntax**

```sql
sa_get_server_messages( first_line )
```

**Arguments**

- **first_line**  Use this INTEGER parameter to specify the line number from which to start displaying server messages.
### Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line_num</td>
<td>INTEGER</td>
<td>The line number of a server message.</td>
</tr>
<tr>
<td>message_text</td>
<td>VARCHAR(255)</td>
<td>The server message text.</td>
</tr>
<tr>
<td>message_time</td>
<td>TIMESTAMP</td>
<td>The time of the message.</td>
</tr>
</tbody>
</table>

### Remarks

This procedure takes an INTEGER parameter that specifies the starting line number to display, and returns a row for that line and for all subsequent lines. If the starting line is negative, the result set starts at the first available line. The result set includes the line number, message text, and message time.

### Privileges

None

### Side effects

None

### Example

The following example uses the `sa_get_server_messages` system procedure to return a result set containing the content of the database server messages window, starting from line 16.

```sql
CALL sa_get_server_messages( 16 );
```

<table>
<thead>
<tr>
<th>line_num</th>
<th>message_text</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Running on Windows XP Build 2195...</td>
</tr>
<tr>
<td>17</td>
<td>2132K of memory used for caching</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### sa_get_table_definition system procedure

Returns a LONG VARCHAR string containing the SQL statements required to create the specified table and its indexes, foreign keys, triggers, and granted privileges.

### Syntax

```sql
sa_get_table_definition(
    @owner    ,
    @tabname
)
```
Arguments

- **@owner** Use this VARCHAR(128) parameter to specify the owner of tabname.
- **@tabname** Use this VARCHAR(128) parameter to specify the name of the table.

Remarks

This function returns a LONG VARCHAR string containing the SQL statements required to create the specified table and its indexes, foreign keys, triggers, and granted privileges. To create a new table with the same definition, use the string returned by this function with the EXECUTE IMMEDIATE statement and the LOCATE, SUBSTRING, and REPLACE functions.

Privileges

You must have the SELECT ANY TABLE system privilege, or SELECT privilege on the SYSUSERPERM compatibility view.

Side effects

None

See also

- “SYSUSERPERM compatibility view (deprecated)” on page 1447
- “sa_split_list system procedure” on page 1248
- “EXECUTE IMMEDIATE statement [SP]” on page 791
- “LOCATE function [String]” on page 287
- “SUBSTRING function [String]” on page 379
- “REPLACE function [String]” on page 345

Example

The following statement uses the sa_get_table_definition function to display the string containing the SQL statements required to create the Departments table.

```sql
SELECT sa_get_table_definition( 'GROUPO', 'Departments');
```

**sa_get_user_status system procedure**

Allows you to determine the current status of users.

Syntax

```sql
sa_get_user_status( )
```

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique number identifying the user.</td>
</tr>
</tbody>
</table>
### Column name | Data type | Description
--- | --- | ---
user_name | CHAR(128) | The name of the user.
connections | INTEGER | The current number of connections by this user.
failed_logins | UNSIGNED INTEGER | The number of failed login attempts made by the user.
last_login_time | TIMESTAMP | The local time that the user last logged in.
locked | TINYINT | Indicates if the user account is locked.
reason_locked | LONG VARCHAR | The reason the account is locked.
user_dn | CHAR(1024) | The Distinguished Name (DN) for a user ID connecting to an LDAP server.
user_dn_cached_at | TIMESTAMP | The local time that the DN was stored.
password_change_state | BIT | A value that indicates whether a dual password change is in progress (0=No, 1=Yes). The default is 0.
password_change_first_user | UNSIGNED INTEGER | The user_id of the user who set the first part of a dual password; otherwise NULL.
password_change_second_user | UNSIGNED INTEGER | The user_id of the user who set the second part of a dual password; otherwise NULL.

**Remarks**
This procedure returns a result set that shows the current status of users. In addition to basic user information, the procedure includes a column indicating if the user has been locked out and a column with a reason for the lockout. Users can be locked out for the following reasons: locked due to policy, password expiry, or too many failed attempts.

**Privileges**
You can view information about yourself; no privilege is required. You must have the MANAGE ANY USER system privilege to view information about other users.

**Side effects**
None
See also

- “Login policies” [SQL Anywhere Server - Database Administration]
- “Creating a new login policy” [SQL Anywhere Server - Database Administration]
- “Creating a user (Sybase Central)” [SQL Anywhere Server - Database Administration]
- “Assigning a login policy to an existing user” [SQL Anywhere Server - Database Administration]
- “Altering a login policy” [SQL Anywhere Server - Database Administration]
- “Deleting a login policy” [SQL Anywhere Server - Database Administration]
- “Dual control passwords” [SQL Anywhere Server - Database Administration]

Example

The following example uses the `sa_get_user_status` system procedure to return the status of database users.

```sql
CALL sa_get_user_status;
```

### sa_http_header_info system procedure

Returns HTTP request header names and values.

**Syntax**

```sql
sa_http_header_info( [header_parm] )
```

**Arguments**

- `header_parm`  Use this optional VARCHAR(255) parameter to specify an HTTP header name. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>VARCHAR(255)</td>
<td>The HTTP header name.</td>
</tr>
<tr>
<td>Value</td>
<td>LONG VARCHAR</td>
<td>The HTTP header value.</td>
</tr>
</tbody>
</table>

**Remarks**

The `sa_http_header_info` system procedure returns header names and values. If you do not specify the header name using the optional parameter, the result set contains values for all headers.

This procedure returns a non-empty result set if it is called while processing an HTTP request within a web service.

**Privileges**

None

**Side effects**

None
See also

- “NEXT_HTTP_HEADER function [Web service]” on page 306
- “HTTP_HEADER function [Web service]” on page 266
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Example

The following web service procedure which is called from a web service illustrates the use of the sa_http_header_info system procedure.

```sql
CREATE OR REPLACE PROCEDURE User1.HTTPHeaderExample()
RESULT ( html_string LONG VARCHAR )
BEGIN
  DECLARE myname VARCHAR(255);
  DECLARE myvalue LONG VARCHAR;
  DECLARE err_notfound
    EXCEPTION FOR SQLSTATE '02000';
  DECLARE curs CURSOR FOR
    SELECT Name, Value FROM sa_http_header_info();
  MESSAGE '=== HTTP Headers ===' TO CONSOLE;
  OPEN curs;
  FetchLoop: LOOP
    FETCH next curs INTO myname, myvalue;
    IF SQLSTATE = err_notfound THEN
      LEAVE FetchLoop;
    END IF;
    MESSAGE myname, '=', myvalue TO CONSOLE;
  END LOOP FetchLoop;
  CLOSE curs;
END;
```

When the web service that calls this web service procedure is used, output appears in the database server messages window that is similar to the following.

```
=== HTTP Headers ===
@HttpQueryString=param1=value1&param2=value2&param3=value3
User-Agent=Mozilla/5.0 (Windows NT 6.1; WOW64; rv:16.0) Gecko/20100101
  Firefox/16.0
Authorization=Basic VXNlcjE6dXNlcg==
Cache-Control=max-age=0
Connection=keep-alive
Host=schueler-t3500.sybase.com:8082
@HttpURI=/ShowHTTPHeaders?param1=value1&param2=value2&param3=value3
@HttpMethod=GET
Accept=text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
@HttpVersion=HTTP/1.1
Accept-Language=en-US,en;q=0.5
Accept-Encoding=gzip, deflate
```

**sa_http_php_page system procedure**

Returns the result of passing the PHP code that is to be interpreted through a PHP interpreter using the current HTTP request for context information such as headers, GET/POST data, protocol version, request URL, method, and so on.
Syntax

```
sa_http_php_page( php_page )
```

Arguments

- **php_page** This LONG VARCHAR parameter contains the entire PHP code that is to be interpreted, including the starting and ending markers (<?php and ?>).

Returns

This function returns a LONG BINARY value.

Remarks

To use this system procedure, the PHP external environment must already be installed.

The owner of this system procedure is DBO. However, for improved security, the `sa_http_php_page` system procedure is executed as the invoker.

Privileges

None

Side effects

None

See also

- “sa_http_php_page_interpreted system procedure” on page 1156
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Example

The following example submits the phpinfo() query to a PHP interpreter and displays the HTML result.

```
SELECT CAST( sa_http_php_page('<?php phpinfo(); ?>') AS LONG VARCHAR );
```

The following example submits a PHP script to a PHP interpreter and displays the XML result.

```
SELECT CAST( sa_http_php_page('<?php
c $conn = sasql_connect( "UID=DBA;PWD=sql" );
 $result = sasql_query( $conn, "SELECT * FROM Employees" );
 sasql_result_all( $result );
 sasql_free_result( $result );
 sasql_disconnect( $conn );
 ?>' ) AS LONG VARCHAR );
```

**sa_http_php_page_interpreted system procedure**

Returns the result of passing the PHP code that is to be interpreted through a PHP interpreter using the specified parameters for context information such as headers, GET/POST data, protocol version, request URL, method, and so on.
Syntax

\[
\text{sa_http_php_page_interpreted(}
\text{php\_page, method, url, version, headers, request\_body)}
\]

Arguments

- **php_page**  This LONG VARCHAR parameter contains the entire PHP code that is to be interpreted, including the starting and ending markers (`<?php` and `?>`).

- **method**  This LONG VARCHAR parameter contains the HTTP request method (for example, GET, POST, PUT, or one of the other standard request methods). The value for method can be determined using the value of @HttpMethod in the current HTTP request.

- **url**  This LONG VARCHAR parameter contains the full HTTP request URL, including the query string, if present. The value for url can be determined using the value of @HttpURI in the current HTTP request.

- **version**  This LONG VARCHAR parameter contains the HTTP request protocol version (for example, HTTP/1.1). The value for version can be determined using the value of @HttpVersion in the current HTTP request.

- **headers**  This LONG BINARY parameter contains the HTTP request headers in the standard HTTP header format: `Field-Name: Value\n`. The value for headers can be retrieved from the current HTTP request using the following SELECT statement:

\[
\text{SELECT LIST( name || ': ' || value, CHAR(13) || CHAR(10) )}
\text{FROM sa_http_header_info();}
\]

- **request_body**  This LONG BINARY parameter contains the HTTP request body in binary form. The value for request_body can be retrieved from the current HTTP request using the HTTP_BODY function.

Returns

This function returns a LONG BINARY value.

Remarks

To use this system procedure, the PHP external environment must already be installed.

To use this system procedure outside web services requests, you must provide request information. Any headers set within the PHP code are lost.

The owner of this system procedure is DBO. However, for improved security, the sa_http_php_page_interpreted system procedure is executed as the invoker.
Privileges
None

Side effects
None

See also
- “HTTP_BODY function [Web service]” on page 263
- “The PHP external environment” [SQL Anywhere Server - Programming]
- “sa_http_php_page system procedure” on page 1155
- “sa_http_header_info system procedure” on page 1154
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Example
The following example submits the phpinfo() query to a PHP interpreter and displays the HTML result.

```
BEGIN
  DECLARE headers LONG VARCHAR;
  SELECT list( name || ': ' || value, char(13) || char(10) ) INTO headers
  FROM sa_http_header_info();
  SELECT CAST( sa_http_php_page_interpreted( '<?php phpinfo(); ?>',
      http_header( '@HttpMethod' ),
      http_header( '@HttpURI' ),
      http_header( '@HttpVersion' ),
      headers,
      HTTP_BODY() ) AS LONG VARCHAR);
END;
```

**sa_http_variable_info system procedure**

Returns HTTP variable names and values.

Syntax
```
sa_http_variable_info( [variable_parm] )
```

Arguments
- `variable_parm` Use this optional VARCHAR(255) parameter to specify an HTTP variable name. The default is NULL.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>VARCHAR(255)</td>
<td>The HTTP variable name.</td>
</tr>
<tr>
<td>Value</td>
<td>LONG VARCHAR</td>
<td>The HTTP variable value.</td>
</tr>
</tbody>
</table>
Remarks
The sa_http_variable_info system procedure returns variable names and values. If you do not specify the variable name using the optional parameter, the result set contains values for all variables.

This procedure returns a non-empty result set if it is called while processing an HTTP request within a web service.

Privileges
None

Side effects
None

See also
- “NEXT_HTTP_VARIABLE function [Web service]” on page 309
- “HTTP_VARIABLE function [Web service]” on page 270
- “sa_http_header_info system procedure” on page 1154
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Example
The following web service procedure which is called from a web service illustrates the use of the sa_http_variable_info system procedure.

```
CREATE OR REPLACE PROCEDURE User1.HTTPVariableExample(
RESULT ( html_string LONG VARCHAR )
BEGIN
  DECLARE myname VARCHAR(255);
  DECLARE myvalue LONG VARCHAR;
  DECLARE err_notfound
    EXCEPTION FOR SQLSTATE '02000';
  DECLARE curs CURSOR FOR
      SELECT Name, Value FROM sa_http_variable_info();
  MESSAGE '=== HTTP Variables ===' TO CONSOLE;
  OPEN curs;
  FetchLoop: LOOP
    FETCH next curs INTO myname, myvalue;
    IF SQLSTATE = err_notfound THEN
      LEAVE FetchLoop;
    END IF;
    MESSAGE myname, '=', myvalue TO CONSOLE;
  END LOOP FetchLoop;
  CLOSE curs;
END;
```

For a URL parameter list like ?param1=value1&param2=value2&param3=value3, output from this sample web service procedure appears in the database server messages window as a list in the form of parameter name=value.

```
=== HTTP Variables ===
param1=value1
param3=value3
param2=value2
```
**sa_index_density system procedure**

Reports information about the amount of fragmentation and skew within indexes.

**Syntax**

```sql
sa_index_density(
    [ tbl_name
    [, owner_name ] ]
)
```

**Arguments**

- **tbl_name** Use this optional CHAR(128) parameter to specify the table name. The default is NULL.
- **owner_name** Use this optional CHAR(128) parameter to specify the owner name. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>CHAR(128)</td>
<td>The name of a table.</td>
</tr>
<tr>
<td>TableId</td>
<td>UNSIGNED INTEGER</td>
<td>The table ID.</td>
</tr>
<tr>
<td>IndexName</td>
<td>CHAR(128)</td>
<td>The name of an index.</td>
</tr>
<tr>
<td>IndexId</td>
<td>UNSIGNED INTEGER</td>
<td>The index ID. This column contains one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0 for primary keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SYSFK_.foreign_key_id for foreign keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SYSIDX_.index_id for all other indexes</td>
</tr>
<tr>
<td>IndexType</td>
<td>CHAR(4)</td>
<td>The index type. This column contains one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PKEY for primary keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FKEY for foreign keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- UI for unique indexes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- UC for unique constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- NUI for non-unique indexes</td>
</tr>
</tbody>
</table>
### Column name | Data type | Description
--- | --- | ---
LeafPages | UNSIGNED INTEGER | The number of leaf pages.
Density | DOUBLE | A fraction between 0 and 1 that provides an indication of how full each index page is (on average).
Skew | DOUBLE | A number that provides an indication of the level of unbalance in an index. A value of 1 indicates a perfectly balanced index. Larger values indicate a higher degree of skew.

### Remarks
Use the `sa_index_density` system procedure to obtain information about the degree of fragmentation and skew in indexes. For indexes with a high number of leaf pages, higher density values and lower skew values are desirable.

Index density reflects the average fullness of the index pages, as a percentage. A density of 0.7 indicates that index pages are, on average, 70% full with index data. Index skew reflects the typical deviation from the average density. The amount of skew is important to the optimizer when making selectivity estimates.

When the number of leaf pages is low, you do not need to be concerned about density and skew values. Density and skew values become important only when the number of leaf pages are high. When the number of leaf pages is high, a low density value can indicate fragmentation, and a high skew value can indicate that indexes are not well balanced. Both of these can be factors in poor performance. Executing a `REORGANIZE TABLE` statement addresses both of these issues.

If you do not specify a table when calling this procedure, the information for all indexes on all tables in the database is returned.

You can also use the **Application Profiling Wizard** to determine whether index density and skew are at acceptable levels.

### Privileges
You must have one of the following system privileges:

- MONITOR
- MANAGE ANY STATISTICS
- CREATE ANY INDEX
- ALTER ANY INDEX
- DROP ANY INDEX
- CREATE ANY OBJECT
- ALTER ANY OBJECT
- DROP ANY OBJECT

### Side effects
None
See also

- “Reduce index fragmentation and skew” [SQL Anywhere Server - SQL Usage]
- “REORGANIZE TABLE statement” on page 935
- “Using the Application Profiling Wizard” [SQL Anywhere Server - SQL Usage]

Example

The following example uses the sa_index_density system procedure to return a result set summarizing the amount of fragmentation and skew within all the indexes in the database.

```sql
CALL sa_index_density();
```

**sa_index_levels system procedure**

Assists in performance tuning by reporting the number of levels in an index.

Syntax

```sql
sa_index_levels(
  [ tbl_name ]
  [, owner_name ]
)
```

Arguments

- **tbl_name** Use this optional CHAR(128) parameter to specify the table name. The default is NULL.
- **owner_name** Use this optional CHAR(128) parameter to specify the owner name. The default is NULL.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>CHAR(128)</td>
<td>The name of a table.</td>
</tr>
<tr>
<td>TableId</td>
<td>UNSIGNED INTEGER</td>
<td>The table ID.</td>
</tr>
<tr>
<td>IndexName</td>
<td>CHAR(128)</td>
<td>The name of an index.</td>
</tr>
<tr>
<td>IndexId</td>
<td>UNSIGNED INTEGER</td>
<td>The index ID. This column contains one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0 for primary keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SYSFK.foreign_key_id for foreign keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SYSIDX.index_id for all other indexes</td>
</tr>
</tbody>
</table>
### Remarks

The number of levels in the index tree determines the number of I/O operations needed to access a row using the index. Indexes with a few levels are more efficient than indexes with a large number of levels.

The procedure returns a result set containing the table name, the table ID, the index name, the index ID, the index type, and the number of levels in the index.

If no arguments are supplied, levels are returned for all indexes in the database. If only `tbl_name` is supplied, levels for all indexes on that table are supplied. If `tbl_name` is NULL and an `owner_name` is given, only levels for indexes on tables owned by that user are returned.

### Privileges

You must have one of the following system privileges:

- MANAGE ANY STATISTICS
- CREATE ANY INDEX
- ALTER ANY INDEX
- DROP ANY INDEX
- CREATE ANY OBJECT
- ALTER ANY OBJECT
- DROP ANY OBJECT

### Side effects

None

### See also

- “CREATE INDEX statement” on page 599
- “Proper selection of indexes can make a large performance difference” [SQL Anywhere Server - SQL Usage]
Example

The following example uses the sa_index_levels system procedure to return the number of levels in the Products index.

```
CALL sa_index_levels( );
```

<table>
<thead>
<tr>
<th>TableName</th>
<th>TableId</th>
<th>IndexName</th>
<th>...</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Products</td>
<td>436</td>
<td>Products</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

sa_install_feature system procedure

Installs additional features, for example additional spatial features.

Syntax

```
sa_install_feature( feat_name )
```

Arguments

- **feat_name** A LONG VARCHAR parameter that identifies the feature to install. The default is NULL. The supported feature names are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>st_geometry_predefined_uom</td>
<td>Installs predefined units of measure that are not installed by default in new databases.</td>
</tr>
<tr>
<td>st_geometry_predefined_srs</td>
<td>Installs predefined spatial reference systems and units of measure that are not installed by default in new databases.</td>
</tr>
<tr>
<td>st_geometry_compat_func</td>
<td>Installs a set of spatial compatibility functions. These functions can be used as an alternative to the spatial methods.</td>
</tr>
</tbody>
</table>

Feature name definitions are provided in the st_geometry_config.tgz file located in the %SQLANY16% \scripts directory. If the file is removed and you attempt to install features that are dependent on the file, an error is returned.

Remarks

You can query the feat_name value to see what will be installed. For example, the following query returns the units of measure that would be installed for st_geometry_predefined_uom.

```
SELECT * FROM st_geometry_predefined_uom( 'CREATE' );
```

The previous example also shows you parameter names so you can query for specific values using a WHERE clause. For example, the following statement queries the unit_name parameter for the chain unit of measure:
SELECT * FROM st_geometry_predefined_uom( 'CREATE' ) WHERE unit_name='chain';

| unit_name | unit_type | conversion_factor | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>chain</td>
<td>LINEAR</td>
<td>20.1168</td>
<td>...</td>
</tr>
</tbody>
</table>

The following returns all units of measure that are based on foot:

SELECT * FROM st_geometry_predefined_uom() WHERE unit_name LIKE '%foot%';

Use the following query to find the spatial reference systems that would be installed:

SELECT * FROM st_geometry_predefined_srs();

The following statement queries for a spatial reference system by organization and organization_coordsys_id:

SELECT * FROM st_geometry_predefined_srs() WHERE organization='EPSG' AND organization_coordsys_id=2295;

Privileges

For st_geometry_predefined_uom and st_geometry_predefined_srs, you must have the MANAGE ANY SPATIAL OBJECT system privilege.

For st_geometry_compat_func, you must have the MANAGE ANY OBJECT PRIVILEGE, CREATE ANY PROCEDURE, and SELECT ANY TABLE system privileges.

For sa_install_feature, you must have the MANAGE ANY SPATIAL OBJECT system privilege.

See also

- “Spatial reference systems (SRS) and Spatial reference identifiers (SRID)” [SQL Anywhere Server - Spatial Data Support]
- “Units of measure” [SQL Anywhere Server - Spatial Data Support]
- “Spatial compatibility functions” [SQL Anywhere Server - Spatial Data Support]

Example

The following statement installs all of the predefined units of measure that are not installed by default in a new database:

CALL sa_install_feature( 'st_geometry_predefined_uom' );

The following statement installs a set of spatial compatibility functions which can be used as an alternative to the spatial methods:

CALL sa_install_feature( 'st_geometry_compat_func' );

sa_java_loaded_classes system procedure

Lists the classes currently loaded by the database server into a Java VM.
Syntax

```sql
sa_java_loaded_classes()
```

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class_name</td>
<td>VARCHAR(512)</td>
<td>The name of a class currently loaded by the database server into a Java VM.</td>
</tr>
</tbody>
</table>

Remarks

Returns a result set containing all the names of the Java classes currently loaded by the database server into a Java VM.

The procedure can be useful to diagnose missing classes. It can also be used to identify which classes from a particular jar are used by a given application.

Privileges

None

Side effects

None

See also

- “How to install Java classes into a database” [SQL Anywhere Server - Programming]

Example

The following example calls the init cover function to load and call the init method of the sample Invoice Java class (see “Tutorial: Using Java in the database” [SQL Anywhere Server - Programming]). It then calls the `sa_java_loaded_classes` to obtain a list of all loaded Java classes.

```sql
CALL init( 'Work boots', 79.99, 'Hay fork', 37.49 );
CALL sa_java_loaded_classes();
```

**sa_list_cursors system procedure**

Returns the list of cursors in use by the current connection.

Syntax

```sql
sa_list_cursors()
```
### Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>UNSIGNED INTEGER</td>
<td>A unique handle identifying the cursor.</td>
</tr>
<tr>
<td>scope</td>
<td>INTEGER</td>
<td>The scope of the call stack where the cursor is open.</td>
</tr>
<tr>
<td>cursor_name</td>
<td>VARCHAR(128)</td>
<td>The cursor name.</td>
</tr>
<tr>
<td>is_open</td>
<td>BIT</td>
<td>The indicator of whether the cursor is currently open (1).</td>
</tr>
<tr>
<td>is_pinned</td>
<td>BIT</td>
<td>The indicator of whether the cursor is currently pinned in memory (1) in anticipation of reuse.</td>
</tr>
<tr>
<td>fetch_count</td>
<td>UNSIGNED BIGINT</td>
<td>The number of rows that have been fetched from the cursor.</td>
</tr>
</tbody>
</table>

### Remarks

The `sa_list_cursors` system procedure can be used in a CALL statement or in the FROM clause of a SELECT statement.

### Privileges

None

### Side effects

None

### See also

- “sa_copy_cursor_to_temp_table system procedure” on page 1115
- “sa_describe_cursor system procedure” on page 1124

### Example

The following example returns the list of open cursors for the connection:

```sql
CALL sa_list_cursors();
```

### sa_load_cost_model system procedure

Replaces the current cost model with the cost model stored in the specified file.

#### Syntax

```
sa_load_cost_model( file_name )
```

#### Arguments

- `file_name` Use this CHAR(1024) parameter to specify the name of the cost model file to load.
Remarks

The optimizer uses cost models to determine optimal access plans for queries. The database server maintains a cost model for each database. The cost model for a database can be recalibrated at any time using the CALIBRATE SERVER clause of the ALTER DATABASE statement. For example, you might decide to recalibrate the cost model if you move the database onto non-standard hardware.

The sa_load_cost_model system procedure allows you to load a cost model that has been saved to file (file_name). Loading a cost model replaces the current cost model for the database.

Note

The sa_unload_cost_model system procedure does not include CALIBRATE PARALLEL READ information in the file that sa_load_cost_model loads.

Using the sa_load_cost_model system procedure can eliminate repetitive, time-consuming recalibration activities when there is a large number of identical hardware installations.

Exclusive use of the database is required when loading the new cost model.

When loading a cost model, consider whether it was generated for a database that is located on similar hardware. Loading a cost model from a database that is stored on significantly different hardware may cause poor performance due to inefficient access plans.

Cost models are saved to file using the sa_unload_cost_model system procedure.

Privileges

You must have the ALTER DATABASE and the LOAD ANY TABLE system privileges.

Side effects

The database server performs a COMMIT after loading the new cost model.

See also

- “ALTER DATABASE statement” on page 426
- “sa_unload_cost_model system procedure” on page 1262
- “Advanced: Query optimization” [SQL Anywhere Server - SQL Usage]

Example

The following example loads the cost model from a file called costmodel8:

```sql
CALL sa_load_cost_model( 'costmodel8' );
```

sa_locks system procedure

Displays all locks in the database.

Syntax

```sql
sa_locks(
    [ connection ]
)
```
Arguments

- **connection** Use this INTEGER parameter to specify a connection ID number. The procedure returns lock information only about the specified connection. The default value is 0 (or NULL), in which case information is returned about all connections.

- **creator** Use this CHAR(128) parameter to specify a user ID. The procedure returns information only about the tables owned by the specified user. The default value for the creator parameter is NULL. When this parameter is set to NULL, sa_locks returns the following information:
  - if the table_name parameter is unspecified, locking information is returned for all tables in the database
  - if the table_name parameter is specified, locking information is returned for tables with the specified name that were created by the current user

- **table_name** Use this CHAR(128) parameter to specify a table name. The procedure returns information only about the specified tables. The default value is NULL, in which case information is returned about all tables.

- **max_locks** Use this INTEGER parameter to specify the maximum number of locks for which to return information. The default value is 1000. The value -1 means return all lock information.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conn_name</td>
<td>VARCHAR(128)</td>
<td>The name of the current connection.</td>
</tr>
<tr>
<td>conn_id</td>
<td>INTEGER</td>
<td>The connection ID number.</td>
</tr>
<tr>
<td>user_id</td>
<td>CHAR(128)</td>
<td>The user ID for the connection.</td>
</tr>
<tr>
<td>table_type</td>
<td>CHAR(6)</td>
<td>The type of table. This type is either BASE for a table, GLBTMP for global temporary table, or MVIEW for a materialized view.</td>
</tr>
<tr>
<td>creator</td>
<td>VARCHAR(128)</td>
<td>The owner of the table.</td>
</tr>
<tr>
<td>table_name</td>
<td>VARCHAR(128)</td>
<td>The table on which the lock is held.</td>
</tr>
<tr>
<td>index_id</td>
<td>INTEGER</td>
<td>The index ID or NULL.</td>
</tr>
<tr>
<td>lock_class</td>
<td>CHAR(8)</td>
<td>The lock class. One of Schema, Row, Table, or Position.</td>
</tr>
</tbody>
</table>
## System procedures

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock_duration</td>
<td>CHAR(11)</td>
<td>The duration of the lock. One of Transaction, Position, or Connection.</td>
</tr>
<tr>
<td>lock_type</td>
<td>CHAR(9)</td>
<td>The lock type (this is dependent on the lock class).</td>
</tr>
<tr>
<td>row_identifier</td>
<td>UNSIGNED BIGINT</td>
<td>The identifier for the row. This is either an 8-byte row identifier or NULL.</td>
</tr>
</tbody>
</table>

### Remarks

The `sa_locks` procedure returns a result set containing information about all the locks in the database.

The value in the lock_type column depends on the lock classification in the lock_class column. The following values can be returned:

<table>
<thead>
<tr>
<th>Lock class</th>
<th>Lock types</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>Shared (shared schema lock)</td>
<td>For schema locks, the row_identifier and index ID values are NULL.</td>
</tr>
<tr>
<td></td>
<td>Exclusive (exclusive schema lock)</td>
<td></td>
</tr>
<tr>
<td>Lock class</td>
<td>Lock types</td>
<td>Comments</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Row</td>
<td>Read (read lock)</td>
<td>Row read locks can be short-term locks (scans at isolation level 1) or can be long-term locks at higher isolation levels. The lock_duration column indicates whether the read lock is of short duration because of cursor stability (Position) or long duration, held until COMMIT/ROLLBACK (Transaction). Row locks are always held on a specific row, whose 8-byte row identifier is reported as a 64-bit integer value in the row_identifier column. A surrogate lock is a special case of a row lock. Surrogate locks are held on surrogate entries, which are created when referential integrity checking is delayed. There is not a unique surrogate lock for every surrogate entry created in a table. Rather, a surrogate lock corresponds to the set of surrogate entries created for a given table by a given connection. The row_identifier value is unique for the table and connection associated with the surrogate lock. If required, key and non-key portions of a row can be locked independently. A connection can obtain a read lock on the key portion of a row for shared (read) access so that other connections can still obtain write locks on other non-key columns of a row. Updating non-key columns of a row does not interfere with the insertion and deletion of foreign rows referencing that row.</td>
</tr>
<tr>
<td></td>
<td>Intent (intent lock)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadPK (read lock)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write (write lock)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WriteNoPK (write lock)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Surrogate (surrogate lock)</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>Shared (shared table lock)</td>
<td>See “Table locks” [SQL Anywhere Server - SQL Usage].</td>
</tr>
<tr>
<td></td>
<td>Intent (intent to update table lock)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exclusive (exclusive table lock)</td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>Phantom (phantom lock)</td>
<td>Usually a position lock is also held on a specific row, and that row's 64-bit row identifier appears in the row_identifier column in the result set. However, Position locks can be held on entire scans (index or sequential), in which case the row_identifier column is NULL. A position lock can be associated with a sequential table scan, or an index scan. The index_id column indicates whether the position lock is associated with a sequential scan. If the position lock is held because of a sequential scan, the index_id column is NULL. If the position lock is held as the result of a specific index scan, the index identifier of that index is listed in the index_id column. The index identifier corresponds to the primary key of the ISYSIDX system table, which can be viewed using the SYSIDX view. If the position lock is held for scans over all indexes, the index ID value is -1.</td>
</tr>
<tr>
<td></td>
<td>Insert (insert lock)</td>
<td></td>
</tr>
</tbody>
</table>
Privileges
You must have the MONITOR system privilege.

Side effects
None

See also
- “How locking works” [SQL Anywhere Server - SQL Usage]
- “SYSIDX system view” on page 1364
- “Types of locks” [SQL Anywhere Server - SQL Usage]
- “Schema locks” [SQL Anywhere Server - SQL Usage]
- “Locks during inserts” [SQL Anywhere Server - SQL Usage]
- “Row locks” [SQL Anywhere Server - SQL Usage]
- “Position locks” [SQL Anywhere Server - SQL Usage]

Example
You can execute the following query to identify locks.

    CALL sa_locks();

For another example of this system procedure, and tips to augment the amount of information you can return, see “How to obtain information about locks” [SQL Anywhere Server - SQL Usage].

sa_make_object system procedure
Ensures that a skeletal instance of an object exists before executing an ALTER statement.

Syntax

    sa_make_object(
        objtype,
        objname
        [, owner
        [, tabname ] ]
    )

    objtype: 'procedure' | 'function' | 'view' | 'trigger' | 'service' | 'event'

Arguments
- **objtype** Use this CHAR(30) parameter to specify the type of object being created. If objtype is 'trigger', this argument specifies the owner of the table on which the trigger is to be created.
- **objname** Use this CHAR(128) parameter to specify the name of the object to be created.
● **owner** Use this optional CHAR(128) parameter to specify the owner of the object to be created. The default is NULL.

● **tabname** This CHAR(128) parameter is required only if objtype is 'trigger', in which case you use it to specify the name of the table on which the trigger is to be created. The default is NULL.

**Remarks**

This procedure is useful in scripts that are run repeatedly to create or modify a database schema. A common problem in such scripts is that the first time they are run, a CREATE statement must be executed, but subsequent times an ALTER statement must be executed. This procedure avoids the necessity of querying the system views to find out whether the object exists.

For procedures, functions, views, triggers, you can now use the OR REPLACE clause instead of this system procedure.

To use the procedure, follow it by an ALTER statement that contains the entire object definition.

**Privileges**

You must have the required privileges as follows:

- **Procedures or functions owned by the invoker** CREATE PROCEDURE, CREATE ANY PROCEDURE, or CREATE ANY OBJECT system privilege

- **Procedures or functions owned by other users** CREATE ANY PROCEDURE or CREATE ANY OBJECT system privilege

- **Services** MANAGE ANY WEB SERVICE system privilege

- **Events** MANAGE ANY EVENT or CREATE ANY OBJECT system privilege

- **Views owned by the invoker** CREATE VIEW, CREATE ANY VIEW, or CREATE ANY OBJECT system privilege

- **Views owned by other users** CREATE ANY VIEW or CREATE ANY OBJECT system privilege

- **Triggers** If the trigger is on a table owned by you, you must have either the CREATE ANY TRIGGER or CREATE ANY OBJECT system privilege.

  If the trigger is on a table owned by another user, you must have either the CREATE ANY TRIGGER or the CREATE ANY OBJECT system privilege. Additionally, you must have one of the following:

  ○ ALTER ANY TABLE privilege

  ○ ALTER ANY OBJECT system privilege

  ○ ALTER permission on the table on which the trigger is being created.

**Side effects**

Automatic commit
See also

- “ALTER EVENT statement” on page 436
- “ALTER FUNCTION statement” on page 439
- “ALTER PROCEDURE statement” on page 456
- “ALTER SERVICE statement [HTTP web service]” on page 466
- “ALTER SERVICE statement [SOAP web service]” on page 471
- “ALTER TRIGGER statement” on page 507
- “ALTER VIEW statement” on page 511

Examples

The following statements ensure that a skeleton procedure definition is created, define the procedure, and grant privileges on it. A script file containing these instructions could be run repeatedly against a database without error.

```sql
CALL sa_make_object( 'procedure', 'myproc' );
ALTER PROCEDURE myproc( in p1 INT, in p2 CHAR(30) )
BEGIN
    // ...
END;
GRANT EXECUTE ON myproc TO public;
```

The following example uses the `sa_make_object` system procedure to add a skeleton web service.

```sql
CALL sa_make_object( 'service', 'my_web_service' );
```

### sa_materialized_view_can_be_immediate system procedure

Returns whether the specified materialized view can be defined as immediate.

**Syntax**

```sql
sa_materialized_view_can_be_immediate( 
    view_name, 
    owner_name 
)
```

**Arguments**

- **view_name** Use this CHAR(128) parameter to specify the name of the materialized view. If `view_name` is NULL, an error is returned.

- **owner_name** Use this CHAR(128) parameter to specify the owner of the materialized view. If `owner_name` is NULL, the current user ID is used.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLStateVal</td>
<td>CHAR(6)</td>
<td>The SQLSTATE returned.</td>
</tr>
</tbody>
</table>
### ErrorMessage

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorMessage</td>
<td>LONG VARCHAR</td>
<td>The error message corresponding to the SQLSTATE.</td>
</tr>
</tbody>
</table>

### Remarks

There are restrictions on whether the specified manual view can be changed to an immediate view. Use this system procedure to determine whether the change is permitted.

Each row in the result set corresponds to a single SQLSTATE returned for a view. So, if the materialized view definition violates more than one restriction, the results include multiple rows for the view.

You can combine the output of this system procedure with the output of the sa_materialized_view_info system procedure to get information about the status of views and whether they can be made immediate.

### Privileges

You must be the owner of the materialized view, or have the ALTER ANY MATERIALIZED VIEW or ALTER ANY OBJECT system privilege.

### See also

- “Advanced: Changing the refresh type for a materialized view” [SQL Anywhere Server - SQL Usage]
- “Restrictions when changing a materialized view from manual to immediate” [SQL Anywhere Server - SQL Usage]
- “sa_materialized_view_info system procedure” on page 1176

### Side effects

All metadata for the specified materialized view, and all dependencies, are loaded into the server cache.

### Example

Execute the following statements to create a manual view, view10, and refresh it:

```sql
CREATE MATERIALIZED VIEW view10
    AS (SELECT C.ID, C.Surname, sum(P.UnitPrice) as revenue, C.CompanyName, SO.OrderDate
    FROM Customers C, SalesOrders SO, SalesOrderItems SOI, Products P
    WHERE C.ID = SO.CustomerID
    AND SO.ID = SOI.ID
    AND P.ID = SOI.ProductID
    GROUP BY C.ID, C.Surname, C.CompanyName, SO.OrderDate);

REFRESH MATERIALIZED VIEW view10;
```

Use the following query to find the reasons why view10 cannot be changed to an immediate view:

```sql
SELECT SQLStateVal AS "SQLstate", ErrorMessage AS Description
FROM sa_materialized_view_can_be_immediate( 'view10', NULL )
ORDER BY SQLSTATE;
```
<table>
<thead>
<tr>
<th>SQLstate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>42WC3</td>
<td>See “The materialized view %1 cannot be changed to immediate because it has already been initialized” [Error Messages].</td>
</tr>
<tr>
<td>42WCA</td>
<td>See “The materialized view %1 cannot be changed to immediate because it does not have a unique index on non-nullable columns” [Error Messages].</td>
</tr>
<tr>
<td>42WC6</td>
<td>See “The materialized view cannot be changed to immediate because COUNT(*) must be part of the SELECT list” [Error Messages].</td>
</tr>
<tr>
<td>42WC7</td>
<td>See “The materialized view cannot be changed to immediate because it does not have a unique index on non-aggregate, non-nullable columns” [Error Messages].</td>
</tr>
</tbody>
</table>

**sa_materialized_view_info system procedure**

Returns information about the specified materialized views.

**Syntax**

```sql
sa_materialized_view_info(
  [ view_name
  [, owner_name ] ]
)
```

**Arguments**

- **view_name**  Use this optional CHAR(128) parameter to specify the name of the materialized view for which to return information. The default is NULL.

- **owner_name**  Use this optional CHAR(128) parameter to specify the owner of the materialized view. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner-Name</td>
<td>CHAR(128)</td>
<td>The owner of the view.</td>
</tr>
<tr>
<td>ViewName</td>
<td>CHAR(128)</td>
<td>The name of the view.</td>
</tr>
<tr>
<td>Status</td>
<td>CHAR(1)</td>
<td>Status information about the view. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● D  disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● E  enabled</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DataStatus</td>
<td>CHAR(1)</td>
<td>Status information about data in the view. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>E</strong>  An error occurred during the last refresh attempt. The view is enabled, but uninitialized.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>F</strong>  The underlying tables have not changed since the last refresh, and the view is considered fresh. The view is enabled and initialized.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>N</strong>  The view is uninitialized. This occurs when one of the following is true:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- the view has not been refreshed since it was created</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- the data has been truncated from the view</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- the view is disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>S</strong>  An underlying table has changed since the last refresh, and the view is considered stale. The view is enabled and initialized.</td>
</tr>
<tr>
<td>ViewLastRefreshed</td>
<td>TIME-</td>
<td>The local time when the view was last refreshed. If the value of ViewLastRefreshed is NULL, the view is uninitialized.</td>
</tr>
<tr>
<td></td>
<td>STAMP</td>
<td></td>
</tr>
<tr>
<td>DataLastModified</td>
<td>TIME-</td>
<td>For a stale view, the last local time that underlying data was modified. The value is NULL for views that are not initialized, or for views that are not considered stale.</td>
</tr>
<tr>
<td></td>
<td>STAMP</td>
<td></td>
</tr>
</tbody>
</table>
### Column name  | Data type  | Description
--- | --- | ---
AvailForOptimization  | CHAR(1)  | Information about the availability of the view for use by the optimizer. Possible values are:
  - **D** Use by the optimizer is disabled. The owner of the view doesn't allow the view to be used by the optimizer.
  - **I** The view cannot be used by the optimizer for some internal reason, for example its definition doesn't meet the conditions required. However, the owner has not explicitly disallowed its use by the optimizer.
  - **N** The view contains no data because a refresh has not been done or has failed. The view can be used by the optimizer by the owner of the view, but it is not initialized.
  - **O** There is an incompatible option value for current connection. The view can be used by the optimizer and its definition meets all the required conditions, but the current option settings are not compatible with the options settings used to create the view.
  - **Y** The view can be used by the optimizer. The owner of the view allows the view to be used by the optimizer and the view definition meets all the conditions needed to be used by the optimizer.

RefreshType  | CHAR(1)  | The refresh type for the view. Possible values are:
  - **I** The view is an immediate view. Immediate views are refreshed immediately when changes to the data in an underlying table impact the data in the materialized view.
  - **M** The view is a manual view. Manual views are refreshed manually, for example using the REFRESH MATERIALIZED VIEW statement, or the sa_refresh_materialized_views system procedure.

### Remarks
If neither `view_name` nor `owner_name` are specified or both are NULL, information about all materialized views in the database is returned.

If `owner_name` is not specified or is NULL, information about all materialized views named `view_name` is returned.

This procedure can be useful for determining the list of materialized views that will never be considered by the optimizer because of a problem with the view definition. The AvailForOptimization value is **I** for these materialized views.
The following table shows how the AvailForOptimization property is determined. Starting from the left column, you read across the row to see the conditions that must be in place to result in the value found in the AvailForOptimization column.

<table>
<thead>
<tr>
<th>User allows view to be used in optimization?</th>
<th>The view definition satisfies all the conditions required for use?</th>
<th>The connection options match those required for use of the view?</th>
<th>The view is initialized?</th>
<th>AvailForOptimization value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Y</td>
</tr>
<tr>
<td>No</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>D</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>I</td>
</tr>
<tr>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>No</td>
<td>N</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>O</td>
</tr>
</tbody>
</table>

An initialized materialized view can be empty. This occurs when there is no data in the underlying tables that meets the materialized view definition. An empty view is not considered the same as an uninitialized materialized view, which also has no data in it. The value of the ViewLastRefreshed property allows you to distinguish between whether the view is uninitialized (NULL), or empty because of data in the underlying tables (non-NULL).

**Privileges**

None

**Side effects**

All metadata for the specified materialized views, and all dependencies, are loaded into the database server cache.

**See also**

- “Materialized views” [SQL Anywhere Server - SQL Usage]
- “Advanced: Changing the refresh type for a materialized view” [SQL Anywhere Server - SQL Usage]
- “Restrictions when changing a materialized view from manual to immediate” [SQL Anywhere Server - SQL Usage]
- “sa_materialized_view_can_be_immediate system procedure” on page 1174
- “Determining which materialized views were considered by the optimizer” [SQL Anywhere Server - SQL Usage]
- “Materialized views restrictions” [SQL Anywhere Server - SQL Usage]
- “Whether to set refresh type to manual or immediate” [SQL Anywhere Server - SQL Usage]

**Example**

The following statement returns information about all materialized views in the database:
The results of the `sa_materialized_view_info` system procedure can be combined with the results of the `sa_materialized_view_can_be_immediate` system procedure to return status information, and whether the view is eligible for being an immediate view. Execute the following statements to create materialized views that are examined for this example:

```sql
CREATE MATERIALIZED VIEW view0 AS ( 
    SELECT ID, Name, Description, Size 
    FROM Products 
    WHERE Quantity > 0 );
CREATE UNIQUE INDEX u_view0 
ON view0( ID );
ALTER MATERIALIZED VIEW view0 
IMMEDIATE REFRESH;
CREATE MATERIALIZED VIEW view00 AS ( 
    SELECT ID, Name, Description, Size 
    FROM Products 
    WHERE Quantity <= 0 );
CREATE UNIQUE INDEX u_view00 
ON view00( ID );
CREATE MATERIALIZED VIEW view1 AS ( 
    SELECT ID, Name, Description, Size 
    FROM Products 
    WHERE Quantity = 0 );
ALTER MATERIALIZED VIEW view1 
DISABLE;
CREATE MATERIALIZED VIEW view100 
AS (SELECT C.ID, C.Surname, sum(P.UnitPrice) as revenue, C.CompanyName, 
    SO.OrderDate 
FROM Customers C, SalesOrders SO, SalesOrderItems SOI, Products P 
WHERE C.ID = SO.CustomerID 
AND SO.ID = SOI.ID 
AND P.ID = SOI.ProductID 
GROUP BY C.ID, C.Surname, C.CompanyName, SO.OrderDate); 
REFRESH MATERIALIZED VIEW view100;
```

Execute the following statement to return the status and eligibility information for the views owned by you:

```sql
SELECT ViewName, Status, ViewLastRefreshed, AvailForOptimization, 
    RefreshType, CanBeImmediate 
FROM sa_materialized_view_info() AS V, 
    LATERAL( SELECT LIST( ErrorMessage, '' ) 
FROM sa_materialized_view_can_be_immediate( V.ViewName, 
    V.OwnerName ) ) AS I( CanBeImmediate ) 
WHERE OwnerName = USER_NAME();
```
<table>
<thead>
<tr>
<th>ViewName</th>
<th>Status</th>
<th>ViewLastRefreshed</th>
<th>AvailForOptimization</th>
<th>RefreshType</th>
<th>CanBeImmediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>view0</td>
<td>E</td>
<td>(NULL)</td>
<td>N</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>view00</td>
<td>E</td>
<td>(NULL)</td>
<td>N</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>view1</td>
<td>D</td>
<td>(NULL)</td>
<td>N</td>
<td>M</td>
<td>Cannot use view 'view1' because it has been disabled</td>
</tr>
<tr>
<td>ViewName</td>
<td>Status</td>
<td>ViewLastRefreshed</td>
<td>AvailForOptimization</td>
<td>RefreshType</td>
<td>CanBelImmediate</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>-------------------</td>
<td>----------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>view100</td>
<td>E</td>
<td>2008-02-12 16:47:00.000</td>
<td>Y</td>
<td>M</td>
<td>The materialized view view100 cannot be changed to immediate because it has already been initialized. The materialized view view100 cannot be changed to immediate because it does not have a unique index on non-nullable columns. The materialized view cannot be changed to immediate because COUNT(*) is required to be part of the SELECT list. The materialized view cannot be changed to immediate because it does not have a unique index on non-aggregate non-nullable columns.</td>
</tr>
</tbody>
</table>
From the results you can see that:

- view0 was never refreshed and is an immediate view.
- view00 was never refreshed and is a manual view.
- view1 is disabled
- view100 is a manual view that was last refreshed at 2008-02-12 16:47:00.000.
- view00 can be changed to an immediate view because there are no error messages in the CanBeImmediate column.
- view1 and view100 cannot be changed to immediate views for the reasons listed in the CanBeImmediate column.

**sa_migrate system procedure**

Migrates a set of remote tables to a SQL Anywhere database.

**Syntax**

```sql
sa_migrate(
  base_table_owner
  , server_name
  , table_name
  , owner_name
  , database_name
  , migrate_data
  , drop_proxy_tables
)
```

**Arguments**

- **base_table_owner** Use this VARCHAR(128) parameter to specify the user on the target SQL Anywhere database who owns the migrated tables. Use the GRANT CONNECT statement to create this user. A value is required for this parameter.

- **server_name** Use this VARCHAR(128) parameter to specify the name of the remote server that is being used to connect to the remote database. Use the CREATE SERVER statement to create this server. A value is required for this parameter.

- **table_name** If you are migrating a single table, use this optional VARCHAR(128) parameter to specify the table name. Otherwise, you should specify NULL (the default) for this parameter. Do not specify NULL for both the table_name and owner_name parameters.

- **owner_name** If you are migrating only tables that belong to one owner, use this optional VARCHAR(128) parameter to specify the owner's name. Otherwise, you should specify NULL (the default) for this parameter. Do not specify NULL for both the table_name and owner_name parameters.
- **database_name** Use this optional VARCHAR(128) parameter to specify the name of the remote database. You must specify the database name to migrate tables from only one database on the remote server. Otherwise, use NULL (the default) for this parameter.

- **migrate_data** Use this optional BIT parameter to specify whether the data in the remote tables is migrated. This parameter can be 0 (do not migrate data) or 1 (migrate data). By default, data is migrated (1 is the default).

- **drop_proxy_tables** Use this optional BIT parameter to specify whether the proxy tables created for the migration process are dropped once the migration is complete. This parameter can be 0 (proxy tables are not dropped) or 1 (proxy tables are dropped). By default, the proxy tables are dropped (1 is the default).

- **migrate_fkeys** Use this optional BIT parameter to specify whether the foreign key mappings are migrated. This parameter can be 0 (do not migrate foreign key mappings) or 1 (migrate foreign key mappings). By default, the foreign key mappings are migrated (1 is the default).

**Remarks**
You can use this procedure to migrate tables to SQL Anywhere from a remote Oracle, IBM DB2, Microsoft SQL Server, Adaptive Server Enterprise, or SQL Anywhere database. This procedure allows you to migrate in one step a set of remote tables, including their foreign key mappings, from the specified server. The sa_migrate system procedure calls the following system procedures:

- sa_migrate_create_remote_table_list
- sa_migrate_create_tables
- sa_migrate_data
- sa_migrate_create_remote_fks_list
- sa_migrate_create_fks
- sa_migrate_drop_proxy_tables

You might want to use these system procedures instead of sa_migrate if you need more flexibility. For example, if you are migrating tables with foreign key relationships that are owned by different users, you cannot retain the foreign key relationships if you use sa_migrate.

Before you can migrate any tables, you must first create a remote server to connect to the remote database using the CREATE SERVER statement. You may also need to create an external login to the remote database using the CREATE EXTERNLOGIN statement.

You can migrate all the tables from the remote database to a SQL Anywhere database by specifying only the `base_table_owner` and `server_name` parameters. However, if you specify only these two parameters, all the tables that are migrated will belong to one owner in the target SQL Anywhere database. If tables have different owners on the remote database and you want them to have different owners on the SQL Anywhere database, then you must migrate the tables for each owner separately, specifying the `base_table_owner` and `owner_name` parameters each time you call the sa_migrate procedure.
Caution
Do not specify NULL for both the table_name and owner_name parameters. Supplying NULL for both the table_name and owner_name parameters migrates all the tables in the database, including system tables. As well, tables that have the same name, but different owners in the remote database all belong to one owner in the target database. It is recommended that you migrate tables associated with one owner at a time.

Privileges
You must have the following system privileges:

- CREATE TABLE or CREATE ANY TABLE (if you are not the base table owner)
- SELECT ANY TABLE (if you are not the base table owner)
- INSERT ANY TABLE (if you are not the base table owner)
- ALTER ANY TABLE (if you are not the base table owner)
- CREATE ANY INDEX (if you are not the base table owner)
- DROP ANY TABLE (if you are not the base table owner)

Side effects
None

See also

- “Database migration to SQL Anywhere” [SQL Anywhere Server - SQL Usage]
- “CREATE EXTERNLOGIN statement” on page 578
- “CREATE SERVER statement” on page 657
- “GRANT statement” on page 827
- “sa_migrate_create_remote_table_list system procedure” on page 1188
- “sa_migrate_create_tables system procedure” on page 1189
- “sa_migrate_data system procedure” on page 1191
- “sa_migrate_create_remote_fks_list system procedure” on page 1187
- “sa_migrate_create_fks system procedure” on page 1186
- “sa_migrate_drop_proxy_tables system procedure” on page 1192

Examples
The following statement migrates all the tables belonging to user DBA from the remote database, including foreign key mappings; migrates the data in the remote tables; and drops the proxy tables when migration is complete. In this example, all the tables that are migrated belong to LocalUser in the target SQL Anywhere database.

CALL sa_migrate( 'LocalUser', 'RemoteSA', NULL, 'DBA', NULL, 1, 1, 1 );

The following statement migrates all the tables that belong to user DBA from the remote database. In the target SQL Anywhere database, these tables belong to the user LocalUser. Proxy tables created during the migration are not dropped at completion and belong to LocalUser.
CALL sa_migrate( 'LocalUser', 'RemoteSA', NULL, 'DBA', NULL, 1, 0, 1 );

**sa_migrate_create_fks system procedure**

Creates foreign keys for each table listed in the dbo.migrate_remote_fks_list table.

**Syntax**

```
sa_migrate_create_fks( i_table_owner )
```

**Arguments**

- **i_table_owner** Use this VARCHAR(128) parameter to specify the user on the target SQL Anywhere database who owns the migrated foreign keys. To migrate tables that belong to different user, execute this procedure for each user whose tables you want to migrate. The `i_table_owner` is created using the GRANT CONNECT statement. A value is required for this parameter.

**Remarks**

This procedure creates foreign keys for each table that is listed in the dbo.migrate_remote_fks_list table. The user specified by the `i_table_owner` argument owns the foreign keys in the target database.

If the tables in the target SQL Anywhere database do not all have the same owner, you must execute this procedure for each user who owns tables for which you need to migrate foreign keys.

**Note**

This system procedure is used with several other migration system procedures, which must be executed in sequence as listed below:

1. sa_migrate_create_remote_table_list
2. sa_migrate_create_tables
3. sa_migrate_data
4. sa_migrate_create_remote_fks_list
5. sa_migrate_create_fks
6. sa_migrate_drop_proxy_tables

As an alternative, you can migrate all tables in one step using the sa_migrate system procedure.

**Privileges**

You must have the following system privileges:

- ALTER ANY TABLE
- CREATE ANY INDEX
Side effects

None

See also

- “Database migration to SQL Anywhere” [SQL Anywhere Server - SQL Usage]
- “GRANT statement” on page 827
- “sa_migrate system procedure” on page 1183
- “sa_migrate_create_remote_table_list system procedure” on page 1188
- “sa_migrate_create_tables system procedure” on page 1189
- “sa_migrate_data system procedure” on page 1191
- “sa_migrate_create_remote_fks_list system procedure” on page 1187
- “sa_migrate_drop_proxy_tables system procedure” on page 1192

Example

The first statement creates a list of a list of foreign keys for the tables that are listed in the
dbo.migrate_remote_table_list table. The second statement creates foreign keys based on the
dbo.migrate_remote_fks_list table. The foreign keys belong to tables owned by the user LocalUser on the
local SQL Anywhere database.

```
CALL sa_migrate_create_remote_fks_list( 'RemoteSA' );
CALL sa_migrate_create_fks( 'LocalUser' );
```

**sa_migrate_create_remote_fks_list system procedure**

Populates the dbo.migrate_remote_fks_list table.

Syntax

```
sa_migrate_create_remote_fks_list( server_name )
```

Arguments

- **server_name** Use this VARCHAR(128) parameter to specify the name of the remote server that is
  being used to connect to the remote database. The remote server is created with the CREATE
  SERVER statement. A value is required for this parameter.

Remarks

This procedure populates the dbo.migrate_remote_fks_list table with a list of foreign keys that can be
migrated from the remote database. You can delete rows from this table for foreign keys that you do not
want to migrate.

As an alternative, you can migrate all tables in one step using the sa_migrate system procedure.

This system procedure is used with several other migration system procedures. The note in the Remarks
section of the sa_migrate_create_fks system procedure contains the list of migrate procedures, and the
order in which you must execute them.

Privileges

None
Side effects

None

See also

- “CREATE SERVER statement” on page 657
- “Database migration to SQL Anywhere” [SQL Anywhere Server - SQL Usage]
- “sa_migrate system procedure” on page 1183
- “sa_migrate_create_remote_table_list system procedure” on page 1188
- “sa_migrate_create_tables system procedure” on page 1189
- “sa_migrate_data system procedure” on page 1191
- “sa_migrate_create_fks system procedure” on page 1186
- “sa_migrate_drop_proxy_tables system procedure” on page 1192

Example

The following statement creates a list of foreign keys for the tables that are listed in the dbo.migrate_remote_table_list table.

```sql
CALL sa_migrate_create_remote_fks_list( 'RemoteSA' );
```

sa_migrate_create_remote_table_list system procedure

Populates the dbo.migrate_remote_table_list table.

Syntax

```sql
sa_migrate_create_remote_table_list(
    i_server_name
    [, i_table_name
    [, i_owner_name
    [, i_database_name ]]]
)
```

Arguments

- **i_server_name** Use this VARCHAR(128) parameter to specify the name of the remote server that is being used to connect to the remote database. The remote server is created with the CREATE SERVER statement. A value is required for this parameter.

- **i_table_name** Use this optional VARCHAR(128) parameter to specify the name(s) of the tables that you want to migrate, or NULL to migrate all the tables. The default is NULL. Do not specify NULL for both the `i_table_name` and `i_owner_name` parameters.

- **i_owner_name** Use this optional VARCHAR(128) parameter to specify the user who owns the tables on the remote database that you want to migrate, or NULL to migrate all the tables. The default is NULL. Do not specify NULL for both the `i_table_name` and `i_owner_name` parameters.

- **i_database_name** Use this optional VARCHAR(128) parameter to specify the name of the remote database from which you want to migrate tables. The default is NULL. When migrating tables from Adaptive Server Enterprise and Microsoft SQL Server databases, you must specify the database name.
Remarks
This procedure populates the dbo.migrate_remote_table_list table with a list of tables that can be migrated from the remote database. You can delete rows from this table for remote tables that you do not want to migrate.

If you do not want all the migrated tables to have the same owner on the target SQL Anywhere database, you must execute this procedure for each user whose tables you want to migrate.

As an alternative, you can migrate all tables in one step using the sa_migrate system procedure.

Caution
Do not specify NULL for both the i_table_name and i_owner_name parameters. Supplying NULL for both the i_table_name and i_owner_name parameters migrates all the tables in the database, including system tables. As well, tables that have the same name, but different owners in the remote database all belong to one owner in the target database. It is recommended that you migrate tables associated with one owner at a time.

This system procedure is used with several other migration system procedures. The note in the Remarks section of the sa_migrate_create_fks system procedure contains the list of migrate procedures, and the order in which you must execute them.

Privileges
None

Side effects
None

See also
- “CREATE SERVER statement” on page 657
- “Database migration to SQL Anywhere” [SQL Anywhere Server - SQL Usage]
- “sa_migrate system procedure” on page 1183
- “sa_migrate_create_tables system procedure” on page 1189
- “sa_migrate_data system procedure” on page 1191
- “sa_migrate_create_remote_fks_list system procedure” on page 1187
- “sa_migrate_create_fks system procedure” on page 1186
- “sa_migrate_drop_proxy_tables system procedure” on page 1192

Example
The following statement creates a list of tables that belong to the user DBA on the remote database.

CALL sa_migrate_create_remote_table_list( 'RemoteSA', NULL, 'DBA', NULL );

**sa_migrate_create_tables system procedure**
Creates a proxy table and base table for each table listed in the dbo.migrate_remote_table_list table.
**Syntax**

```
sa_migrate_create_tables( i_table_owner )
```

**Arguments**

- `i_table_owner` Use this VARCHAR(128) parameter to specify the user on the target SQL Anywhere database who owns the migrated tables. This user is created using the GRANT CONNECT statement. A value is required for this parameter.

**Remarks**

This procedure creates a base table and proxy table for each table listed in the `dbo.migrate_remote_table_list` table (created using the `sa_migrate_create_remote_table_list` procedure). These proxy tables and base tables are owned by the user specified by the `i_table_owner` argument. This procedure also creates the same primary key indexes and other indexes for the new table that the remote table has in the remote database.

If you do not want all the migrated tables to have the same owner on the target SQL Anywhere database, you must execute the `sa_migrate_create_remote_table_list` procedure and the `sa_migrate_create_tables` procedure for each user who will own migrated tables.

As an alternative, you can migrate all tables in one step using the `sa_migrate` system procedure.

This system procedure is used with several other migration system procedures. The note in the Remarks section of the `sa_migrate_create_fks` system procedure contains the list of migrate procedures, and the order in which you must execute them.

**Privileges**

You must have the CREATE ANY TABLE system privilege.

**Side effects**

None

**See also**

- “Database migration to SQL Anywhere” [SQL Anywhere Server - SQL Usage]
- “sa_migrate system procedure” on page 1183
- “sa_migrate_create_remote_table_list system procedure” on page 1188
- “sa_migrate_data system procedure” on page 1191
- “sa_migrate_create_remote_fks_list system procedure” on page 1187
- “sa_migrate_create_fks system procedure” on page 1186
- “sa_migrate_drop_proxy_tables system procedure” on page 1192

**Example**

The first statement creates a list of tables that belong to the user DBA on the remote database. The second statement creates base tables and proxy tables on the target SQL Anywhere database using that list. These tables belong to the user LocalUser.

```sql
CALL sa_migrate_create_remote_table_list( 'RemoteSA', NULL, 'DBA', NULL );
CALL sa_migrate_create_tables( 'LocalUser' );
```
**sa_migrate_data system procedure**

Migrates data from the remote database tables to the target SQL Anywhere database.

**Syntax**

```
sa_migrate_data( i_table_owner )
```

**Arguments**

- `i_table_owner` Use this VARCHAR(128) parameter to specify the user on the target SQL Anywhere database who owns the migrated tables. This user is created using the GRANT CONNECT statement. A value is required for this parameter.

**Remarks**

This procedure migrates the data from the remote database to the SQL Anywhere database for tables belonging to the user specified by the `i_table_owner` argument.

When the tables on the target SQL Anywhere database do not all have the same owner, you must execute this procedure for each user whose tables have data that you want to migrate.

As an alternative, you can migrate all tables in one step using the `sa_migrate` system procedure.

This system procedure is used with several other migration system procedures. The note in the Remarks section of the `sa_migrate_create_fks` system procedure contains the list of migrate procedures, and the order in which you must execute them.

**Privileges**

You must have the following system privileges:

- SELECT ANY TABLE
- INSERT ANY TABLE

**Side effects**

None

**See also**

- “Database migration to SQL Anywhere” [SQL Anywhere Server - SQL Usage]
- “sa_migrate system procedure” on page 1183
- “sa_migrate_create_remote_table_list system procedure” on page 1188
- “sa_migrate_create_tables system procedure” on page 1189
- “sa_migrate_create_remote_fks_list system procedure” on page 1187
- “sa_migrate_create_fks system procedure” on page 1186
- “sa_migrate_drop_proxy_tables system procedure” on page 1192

**Example**

The following statement migrates data to the target SQL Anywhere database for tables that belong to the user LocalUser.
CALL sa_migrate_data( 'LocalUser' );

**sa_migrate_drop_proxy_tables system procedure**

Drops the proxy tables that were created for migration purposes.

**Syntax**

```
sa_migrate_drop_proxy_tables( i_table_owner )
```

**Arguments**

- **i_table_owner**  
  Use this VARCHAR(128) parameter to specify the user on the target SQL Anywhere database who owns the proxy tables. This user is created using the GRANT CONNECT statement. A value is required for this parameter.

**Remarks**

This procedure drops the proxy tables that were created for the migration. The user who owns these proxy tables is specified by the `i_table_owner` argument.

If the migrated tables are not all owned by the same user on the target SQL Anywhere database, you must call this procedure for each user to drop all the proxy tables.

As an alternative, you can migrate all tables in one step using the `sa_migrate` system procedure.

This system procedure is used with several other migration system procedures. The note in the Remarks section of the `sa_migrate_create_fks` system procedure contains the list of migrate procedures, and the order in which you must execute them.

**Privileges**

You must have the DROP ANY TABLE object-level privilege.

**Side effects**

None

**See also**

- “Database migration to SQL Anywhere” [SQL Anywhere Server - SQL Usage]
- “sa_migrate system procedure” on page 1183
- “sa_migrate_create_remote_table_list system procedure” on page 1188
- “sa_migrate_create_tables system procedure” on page 1189
- “sa_migrate_data system procedure” on page 1191
- “sa_migrate_create_remote_fks_list system procedure” on page 1187
- “sa_migrate_create_fks system procedure” on page 1186

**Example**

The following statement drops the proxy tables on the target SQL Anywhere database that belong to the user LocalUser.

```
CALL sa_migrate_drop_proxy_tables( 'LocalUser' );
```
sa_mirror_server_status system procedure

Returns the connection status of the current servers and all the servers that are directly or indirectly receiving log pages from the current server. On primary servers, the procedure returns the status of all connected servers.

Syntax

sa_mirror_server_status( )

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_name</td>
<td>CHAR(128)</td>
<td>The name of the server.</td>
</tr>
<tr>
<td>state</td>
<td>CHAR(20)</td>
<td>The connection status of the server. It can be one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• connected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• disconnected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In some circumstances, a disconnected server has a status of connected. When</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this situation occurs, the last_updated time is in the past.</td>
</tr>
<tr>
<td>last_updated</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>The time the server status was last updated.</td>
</tr>
<tr>
<td>load_current</td>
<td>DOUBLE</td>
<td>The amount of work that the database server is currently performing.</td>
</tr>
<tr>
<td>load_last_1_min</td>
<td>DOUBLE</td>
<td>The amount of work that the database server has performed in the previous minute.</td>
</tr>
<tr>
<td>load_last_5_mins</td>
<td>DOUBLE</td>
<td>The amount of work that the database server has performed in the previous 5 minutes.</td>
</tr>
<tr>
<td>load_last_10_mins</td>
<td>DOUBLE</td>
<td>The amount of work that the database server has performed in the previous 10 minutes.</td>
</tr>
<tr>
<td>num_connections</td>
<td>UNSIGNED INTEGER</td>
<td>The number of connections to the database server.</td>
</tr>
<tr>
<td>num_processors</td>
<td>UNSIGNED INTEGER</td>
<td>The number of database server processors.</td>
</tr>
<tr>
<td>log_written</td>
<td>UNSIGNED BIGINT</td>
<td>The latest transaction log position written to disk based on the last update received from the server.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>log_applied</td>
<td>UNSIGNED BIGINT</td>
<td>The last operation from the transaction log that has been applied based on the last update received from the server. This value is the same as the value of the CurrentRedoPos property</td>
</tr>
</tbody>
</table>

Remarks

Each server updates its status and that of its copy nodes every 5 seconds. On mirror servers, the procedure returns the status of any copy nodes that are receiving log pages from the mirror server, but does not return the status of the primary server. The columns with the prefix load represent a computed load on the database server. The value returned represents the database server load, and not the load from other processes. Higher load values indicate that the database server has more work to perform.

The time in the last_updated column is that of the server in the server_name column. The state is accurate for the time returned in the last_updated column.

When the NodeType connection parameter is specified, the database server uses load information to redirect connections. The database server selects the mirror server with the lowest load; if all servers have the same load, the server with the fewest connections is used.

### Note

For database servers running in the cloud, the NodeType connection parameter does not use load information to redirect connections.

If a copy node was shut down at approximately the same time as its parent, the procedure may still report that the copy node is connected for several minutes. However, the last_updated column remains the same, indicating that the copy node has not reported updated status messages and is likely disconnected.

Privileges

None

Side effects

None

See also

- “CurrentRedoPos database property” [SQL Anywhere Server - Database Administration]
- “NodeType (NODE) connection parameter” [SQL Anywhere Server - Database Administration]
- “Database mirroring” [SQL Anywhere Server - Database Administration]
- “CREATE MIRROR SERVER statement” on page 615
- “ALTER MIRROR SERVER statement” on page 452
- “DROP MIRROR SERVER statement” on page 764

Example

The following example returns the connection status of the current servers and all the servers that are directly or indirectly receiving log pages from the current server when mirroring is in effect.
CALL sa_mirror_server_status( );

**sa_nchar_terms system procedure**

Breaks an NCHAR string into terms and returns each term as a row along with its position.

**Syntax**

```sql
sa_nchar_terms(
  text
  [, config_name
  [, owner ]]
)
```

**Arguments**

- **text**  The LONG NVARCHAR string you are parsing.
- **config_name**  Use this optional CHAR(128) parameter to specify the text configuration object to apply when processing the string. The default is 'default_nchar'.
- **owner**  Use this optional CHAR(128) parameter to specify the owner of the text configuration object. The default value is NULL. The current user is assumed if the owner is not specified or if NULL is specified.

**Remarks**

You can use this system procedure to find out how a string is interpreted when the settings for a text configuration object are applied. This can be helpful when you want to know what terms would be dropped during indexing or from a query string.

**Privileges**

None

**Side effects**

None

**See also**

- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Text configuration object concepts and reference” [SQL Anywhere Server - SQL Usage]
- “sa_char_terms system procedure” on page 1095

**Example**

The following statement returns the terms in the NCHAR string, "It's a work-at-home day!", using the default NCHAR text configuration object, default_nchar:

```sql
CALL sa_nchar_terms (N'It''s a work-at-home day!', 'default_nchar', 'sys');
```
**sa_performance_diagnostics system procedure**

Returns a summary of request timing information for all connections when the database server has request timing logging enabled.

**Syntax**

```sql
sa_performance_diagnostics( )
```

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>INTEGER</td>
<td>Returns the connection ID (a number) for the current connection.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Name</td>
<td>VAR-CHAR(255)</td>
<td>Returns the name of the current connection. You can specify a connection name using the ConnectionName (CON) connection parameter. See “ConnectionName (CON) connection parameter” [SQL Anywhere Server - Database Administration]. The following names are used for temporary connections created by the database server:</td>
</tr>
<tr>
<td>Userid</td>
<td>VAR-CHAR(255)</td>
<td>Returns the user ID for the connection.</td>
</tr>
<tr>
<td>DBNumber</td>
<td>INTEGER</td>
<td>Returns the ID number of the database.</td>
</tr>
<tr>
<td>LoginTime</td>
<td>TIMESTAMP</td>
<td>Returns the date and time the connection was established.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Transaction-StartTime</td>
<td>TIMESTAMP</td>
<td>Returns a string containing the time the database was first modified after a COMMIT or ROLLBACK, or an empty string if no modifications have been made to the database since the last COMMIT or ROLLBACK.</td>
</tr>
<tr>
<td>LastReqTime</td>
<td>TIMESTAMP</td>
<td>Returns the time at which the last request for the specified connection started. This property can return an empty string for internal connections, such as events.</td>
</tr>
<tr>
<td>ReqType</td>
<td>VAR-CHAR(255)</td>
<td>Returns the type of the last request. If a connection has been cached by connection pooling, its ReqType value is CONNECT_POOL_CACHE.</td>
</tr>
<tr>
<td>ReqStatus</td>
<td>VAR-CHAR(255)</td>
<td>Returns the status of the request. It can be one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Idle</strong> The connection is not currently processing a request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Unscheduled</strong> The connection has work to do and is waiting for an available database server worker.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>BlockedIO</strong> The connection is blocked waiting for an I/O.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>BlockedContention</strong> The connection is blocked waiting for access to shared database server data structures.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>BlockedLock</strong> The connection is blocked waiting for a locked object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Executing</strong> The connection is executing a request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The values marked with an asterisk (*) are only returned when logging of request timing information has been turned on for the database server using the -zt server option. If request timing information is not being logged (the default), the values are reported as Executing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ReqTimeUnscheduled</td>
<td>DOUBLE</td>
<td>Returns the amount of unscheduled time, or NULL if the -zt option was not specified. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ReqTimeActive</td>
<td>DOUBLE</td>
<td>Returns the amount of time, in seconds, spent processing requests, or NULL if the -zt option was not specified. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ReqTime-BlockIO</td>
<td>DOUBLE</td>
<td>Returns the amount of time, in seconds, spent waiting for I/O to complete, or NULL if the -zt option was not specified. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ReqTime-BlockLock</td>
<td>DOUBLE</td>
<td>Returns the amount of time, in seconds, spent waiting for a lock, or NULL if the -zt option was not specified. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ReqTime-BlockContention</td>
<td>DOUBLE</td>
<td>Returns the amount of time, in seconds, spent waiting for atomic access, or NULL if the RequestTiming server property is set to Off. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ReqCount-Unschedul-ed</td>
<td>INTEGER</td>
<td>Returns the number of times the connection waited for scheduling, or NULL if the -zt option was not specified. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ReqCount-Active</td>
<td>INTEGER</td>
<td>Returns the number of requests processed, or NULL if the RequestTiming server property is set to Off. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ReqCount-BlockIO</td>
<td>INTEGER</td>
<td>Returns the number of times the connection waited for I/O to complete, or NULL if the -zt option was not specified. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ReqCount-BlockLock</td>
<td>INTEGER</td>
<td>Returns the number of times the connection waited for a lock, or NULL if the -zt option was not specified. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ReqCount-BlockContention</td>
<td>INTEGER</td>
<td>Returns the number of times the connection waited for atomic access, or NULL if the -zt option was not specified. See “-zt database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>LastIdle</td>
<td>INTEGER</td>
<td>Returns the number of ticks between requests.</td>
</tr>
<tr>
<td>BlockedOn</td>
<td>INTEGER</td>
<td>Returns zero if the current connection isn’t blocked, or if it is blocked, the connection number on which the connection is blocked because of a locking conflict.</td>
</tr>
<tr>
<td>UncommittedOp</td>
<td>INTEGER</td>
<td>Returns the number of uncommitted operations.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CurrentProcedure</td>
<td>VAR-CHAR(255)</td>
<td>Returns the name of the procedure that a connection is currently executing. If the connection is executing nested procedure calls, the name is the name of the current procedure. If there is no procedure executing, an empty string is returned.</td>
</tr>
<tr>
<td>EventName</td>
<td>VAR-CHAR(255)</td>
<td>Returns the name of the associated event if the connection is running an event handler. Otherwise, an empty string is returned.</td>
</tr>
<tr>
<td>CurrentLineNumber</td>
<td>INTEGER</td>
<td>Returns the current line number of the procedure or compound statement a connection is executing. The procedure can be identified using the CurrentProcedure property. If the line is part of a compound statement from the client, an empty string is returned.</td>
</tr>
<tr>
<td>LastStatement</td>
<td>LONG VAR-CHAR</td>
<td>Returns the most recently prepared SQL statement for the current connection. The LastStatement value is set when a statement is prepared, and is cleared when a statement is dropped. Only one statement string is remembered for each connection. If sa_conn_activity reports a non-empty value for a connection, it is most likely the statement that the connection is currently executing. If the statement had completed, it would likely have been dropped and the property value would have been cleared. If an application prepares multiple statements and retains their statement handles, then the LastStatement value does not reflect what a connection is currently doing. When client statement caching is enabled, and a cached statement is reused, this property returns an empty string.</td>
</tr>
<tr>
<td>LastPlanText</td>
<td>LONG VAR-CHAR</td>
<td>Returns the long text plan of the last query executed on the connection. You control the remembering of the last plan by setting the RememberLastPlan option of the sa_server_option system procedure, or using the -zp server option. See “-zp database server option” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>AppInfo</td>
<td>LONG VARCHAR</td>
<td>Returns information about the client that made the connection. For HTTP connections, this includes information about the browser. For connections using older versions of jConnect or Sybase Open Client, the information may be incomplete. The API value can be DBLIB, ODBC, OLEDB, ADO.NET, iAnywhereJDBC, PHP, PerlDBD, or DBEXPRESS. For more information about the values returned for other types of connections, see “AppInfo (APP) connection parameter” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>LockCount</td>
<td>INTEGER</td>
<td>Returns the number of locks held by the connection.</td>
</tr>
<tr>
<td>SnapshotCount</td>
<td>INTEGER</td>
<td>Returns the number of snapshots associated with the connection.</td>
</tr>
</tbody>
</table>

Remarks

The `sa_performance_diagnostics` system procedure returns a result set consisting of a set of request timing properties and statistics if the server has been told to collect the information. Recording of request timing information must be turned on the database server before calling `sa_performance_diagnostics`. To do this, specify the `-zt` option when starting the database server or execute the following:

```
CALL sa_server_option( 'RequestTiming','ON' );
```

Privileges

You must have the MONITOR system privilege.

Side effects

None

See also

- “-zt database server option” [SQL Anywhere Server - Database Administration]
- “sa_performance_statistics system procedure” on page 1202
- “sa_server_option system procedure” on page 1222

Examples

You can execute the following query to identify connections that have spent a long time waiting for database server requests to complete.

```
SELECT Number, Name,  
      CAST( DATEDIFF( second, LoginTime, CURRENT TIMESTAMP ) AS DOUBLE ) AS T,  
      IF T <> 0 THEN (ReqTimeActive / T) ELSE NULL ENDIF AS PercentActive  
FROM sa_performance_diagnostics()
```
WHERE T > 0 AND PercentActive > 10.0
ORDER BY PercentActive DESC;

Find all requests that are currently executing, and have been executing for more than 60 seconds:

SELECT Number, Name,
       CAST( DATEDIFF( second, LastReqTime, CURRENT_TIMESTAMP ) AS DOUBLE )
       AS ReqTime
FROM sa_performance_diagnostics()
WHERE ReqStatus <> 'IDLE' AND ReqTime > 60.0
ORDER BY ReqTime DESC;

---

**sa_performance_statistics system procedure**

Returns performance statistics for the server, databases, and connections.

**Syntax**

```
sa_performance_statistics( )
```

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBNumber</td>
<td>INTEGER</td>
<td>Returns the ID number of the database. Returns NULL if the property type is Server.</td>
</tr>
<tr>
<td>ConnNumber</td>
<td>INTEGER</td>
<td>Returns the connection ID (a number) for the current connection. Returns NULL if the Type is Server or Database.</td>
</tr>
<tr>
<td>PropNum</td>
<td>INTEGER</td>
<td>Returns the property number.</td>
</tr>
<tr>
<td>PropName</td>
<td>VARCHAR(255)</td>
<td>Returns the property name.</td>
</tr>
<tr>
<td>Value</td>
<td>INTEGER</td>
<td>Returns the property value.</td>
</tr>
</tbody>
</table>

**Remarks**

The `sa_performance_statistics` system procedure returns a result set consisting of a set of performance statistics. The results are a subset of the results you can return using the `PROPERTY`, `DB_PROPERTY`, and `CONNECTION_PROPERTY` functions.

**Privileges**

You must have the MONITOR system privilege.

**Side effects**

None
See also

- “sa_performance_diagnostics system procedure” on page 1196
- “sa_server_option system procedure” on page 1222

Example

The following example unloads all performance statistics to a text file named `dump_stats.txt`:

```sql
UNLOAD
    SELECT CURRENT_TIMESTAMP, *
    FROM sa_performance_statistics()
    TO 'dump_stats.txt'
    APPEND ON;
```

**sa_post_login_procedure system procedure**

Determines whether a user's password is about to expire.

**Syntax**

```
sa_post_login_procedure()
```

**Arguments**

None

**Result set**

The `sa_post_login_procedure` system procedure returns the following:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message_text</td>
<td>VARCHAR(255)</td>
<td>If message_action is 1, message_text returns the message to display. If message_action is 0, message_text is NULL.</td>
</tr>
<tr>
<td>message_action</td>
<td>INTEGER</td>
<td>Whether the password is about to expire (1=yes, 0=no).</td>
</tr>
</tbody>
</table>

**Remarks**

The `sa_post_login_procedure` system procedure is the default setting for the `post_login_procedure` database option.

`sa_post_login_procedure` uses the user's `password_life_time` and `password_grace_time` login policy option values, and the current date and time, to determine whether a user's password is about to expire. If it is, the message to display to the user is returned in the result set.

**Privileges**

None

**Side effects**

None
See also

- “post_login_procedure option” [SQL Anywhere Server - Database Administration]
- “Login policies” [SQL Anywhere Server - Database Administration]

Example

The following example uses sa_post_login_procedure to determine whether the current user's password is about to expire:

```sql
CALL sa_post_login_procedure();
```

**sa_procedure_profile system procedure**

Reports information about the execution time for each line within procedures, functions, events, or triggers that have been executed in a database.

**Syntax**

```sql
sa_procedure_profile(
    [ filename
    [, save_to_file ] ]
)
```

**Arguments**

- `filename` Use this optional LONG VARCHAR parameter to specify the file to which the profiling information should be saved, or from which file it should be loaded. The default is NULL. See the Remarks section below for more about saving and loading the profiling information.

- `save_to_file` Use this optional INTEGER parameter to specify whether to save the profiling information to a file, or load it from a previously stored file. The default is 0.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>CHAR(1)</td>
<td>The type of object. See the Remarks section below for a list of possible object types.</td>
</tr>
<tr>
<td>object_name</td>
<td>CHAR(128)</td>
<td>The name of the stored procedure, function, event, or trigger. If the object_type is C or D, then this is the name of the foreign key for which the system trigger was defined.</td>
</tr>
<tr>
<td>owner_name</td>
<td>CHAR(128)</td>
<td>The object's owner.</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>The table associated with a trigger (the value is NULL for other object types).</td>
</tr>
<tr>
<td>line_num</td>
<td>UNSIGNED INTEGER</td>
<td>The line number within the procedure.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>executions</td>
<td>UNSIGNED INTEGER</td>
<td>The number of times the line has been executed.</td>
</tr>
<tr>
<td>millisecs</td>
<td>UNSIGNED INTEGER</td>
<td>The time to execute the line, in milliseconds.</td>
</tr>
<tr>
<td>percentage</td>
<td>DOUBLE</td>
<td>The percentage of the total execution time required for the specific line.</td>
</tr>
<tr>
<td>foreign_owner</td>
<td>CHAR(128)</td>
<td>The database user who owns the foreign table for a system trigger.</td>
</tr>
<tr>
<td>foreign_table</td>
<td>CHAR(128)</td>
<td>The name of the foreign table for a system trigger.</td>
</tr>
</tbody>
</table>

**Remarks**

This procedure provides the same information as the Profile tab in Sybase Central.

You can use this procedure to:

- **Return detailed procedure profiling information** To do this, you can simply call the procedure without specifying any arguments.

- **Save detailed procedure profiling information to file** To do this, you must include the filename argument and specify 1 for the save_to_file argument.

- **Load detailed procedure profiling information from a previously saved file** To do this, you must include the filename argument and specify 0 for the save_to_file argument. When using the procedure in this way, the loaded file must have been created by the same database as the one from which you are running the procedure; otherwise, the results may be unusable.

Since the result set includes information about the execution times for individual lines within procedures, triggers, functions, and events, and what percentage of the total procedure execution time those lines use, you can use this profiling information to fine-tune slower procedures that may decrease performance.

Before you can profile your database, you must enable profiling.

The object_type column of the result set can be:

- P stored procedure
- F function
- E event
- T trigger
- C ON UPDATE system trigger
ON DELETE system trigger

If you want summary information instead of line by line details for each execution, use the sa_procedure_profile_summary procedure instead.

Privileges

You must have the MONITOR or MANAGE PROFILING system privilege.

You must also have the following privileges.

- SELECT ANY TABLE (when filename is not NULL and save_to_file is 1)
- LOAD ANY TABLE (when filename is not NULL and save_to_file is 0)

Side effects

None

See also

- “sa_server_option system procedure” on page 1222
- “sa_procedure_profile_summary system procedure” on page 1206
- “Enabling procedure profiling” [SQL Anywhere Server - SQL Usage]

Example

The following statement returns the execution time for each line of every procedure, function, event, or trigger that has been executed in the database:

```sql
CALL sa_procedure_profile( );
```

The following statement returns the same detailed procedure profiling information as the example above, and saves it to a file called detailedinfo.txt:

```sql
CALL sa_procedure_profile( 'detailedinfo.txt', 1 );
```

Either of the following statements can be used to load detailed procedure profiling information from a file called detailedinfo.txt:

```sql
CALL sa_procedure_profile( 'detailedinfo.txt', 0 );
CALL sa_procedure_profile( 'detailedinfo.txt' );
```

sa_procedure_profile_summary system procedure

Reports summary information about the execution times for all procedures, functions, events, or triggers that have been executed in a database. This procedure provides the same information for these objects as the Profile tab in Sybase Central.

Syntax

```sql
sa_procedure_profile_summary([ filename ]
```
Arguments

- **filename**  Use this optional LONG VARCHAR parameter to specify the file to which the profiling information is saved, or from which file it should be loaded. The default is NULL. See the Remarks section below for more about saving and loading the profiling information.

- **save_to_file**  Use this optional INTEGER parameter to specify whether to save the summary information to a file, or to load it from a previously saved file. The default is 0.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_type</td>
<td>CHAR(1)</td>
<td>The type of object. See the Remarks section below for a list of possible object types.</td>
</tr>
<tr>
<td>object_name</td>
<td>CHAR(128)</td>
<td>The name of the stored procedure, function, event, or trigger.</td>
</tr>
<tr>
<td>owner_name</td>
<td>CHAR(128)</td>
<td>The object's owner.</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>The table associated with a trigger (the value is NULL for other object types).</td>
</tr>
<tr>
<td>executions</td>
<td>UNSIGNED INTEGER</td>
<td>The number of times each procedure has been executed.</td>
</tr>
<tr>
<td>millisecs</td>
<td>UNSIGNED INTEGER</td>
<td>The time to execute the procedure, in milliseconds.</td>
</tr>
<tr>
<td>foreign_owner</td>
<td>CHAR(128)</td>
<td>The database user who owns the foreign table for a system trigger.</td>
</tr>
<tr>
<td>foreign_table</td>
<td>CHAR(128)</td>
<td>The name of the foreign table for a system trigger.</td>
</tr>
</tbody>
</table>

Remarks

You can use this procedure to:

- **Return current summary information**  To do this, you can simply call the procedure without specifying any arguments.

- **Save current summary information to file**  To do this, you must include the filename argument and specify 1 for the save_to_file argument.

- **Load stored summary information from a file**  To do this, you must include the filename argument and specify 0 for the save_to_file argument. When using the procedure in this way, the loaded file must have been created by the same database as the one from which you are running the procedure; otherwise, the results may be unusable.
Since the procedure returns information about the usage frequency and efficiency of stored procedures, functions, events, and triggers, you can use this information to fine-tune slower procedures to improve database performance.

Before you can profile your database, you must enable profiling.

The object_type column of the result set can be:

- **P** stored procedure
- **F** function
- **E** event
- **T** trigger
- **S** system trigger
- **C** ON UPDATE system trigger
- **D** ON DELETE system trigger

If you want line by line details for each execution instead of summary information, use the `sa_procedure_profile` procedure instead.

**Privileges**

You must have the MONITOR or MANAGE PROFILING system privilege.

You must also have the following privileges.

- **SELECT ANY TABLE** (when `filename` is not NULL and `save_to_file` is 1)
- **LOAD ANY TABLE** (when `filename` is not NULL and `save_to_file` is 0)

**Side effects**

None

**See also**

- “Enabling procedure profiling” [SQL Anywhere Server - SQL Usage]
- “sa_server_option system procedure” on page 1222
- “sa_procedure_profile system procedure” on page 1204

**Example**

The following statement returns the execution time for any procedure, function, event, or trigger that has been executed in the database:

```sql
CALL sa_procedure_profile_summary( );
```

The following statement returns the same summary information as the previous example, and saves it to a file called `summaryinfo.txt`: 
CALL sa_procedure_profile_summary( 'summaryinfo.txt', 1 );

Either of the following statements can be used to load stored summary information from a file called summaryinfo.txt:

CALL sa_procedure_profile_summary( 'summaryinfo'.txt, 0 );
CALL sa_procedure_profile_summary( 'summaryinfo.txt' );

**sa_recompile_views system procedure**

Locates view definitions stored in the catalog that do not have column definitions and causes the column definitions to be created.

**Syntax**

```sql
sa_recompile_views( [ ignore_errors ] )
```

**Arguments**

- **ignore_errors** Use this optional INTEGER parameter to specify whether to return errors during the recompilation. If you specify 0, an error is returned for each view for which column definition failed. If you specify 1, or any value other than 0, no errors are returned. The default is 0.

**Remarks**

This procedure is used to locate views in the catalog that do not have column definitions and execute an ALTER VIEW statement with the RECOMPILE clause to create the column definitions. The procedure does this for each view that does not have a column definition until there are none left that require compilation or until any remaining column definitions cannot be created. If the procedure is unable to recompile any views, an error is reported. Errors can be suppressed by specifying a non-zero parameter to this procedure.

**Caution**

The sa_recompile_views system procedure should only be called from within a `reload.sql` script. This procedure is used by the Unload utility (dbunload) and should not be used explicitly.

The sa_recompile_views system procedure does not attempt to recompile materialized views or any view marked DISABLED.

**Privileges**

You must have the ALTER ANY VIEW system privilege.

**Side effects**

For each regular view that does not have a VALID status, an ALTER VIEW `owner.viewname` ENABLE statement is executed, causing an automatic commit.
See also

- “Statuses for regular views” [SQL Anywhere Server - SQL Usage]
- “force_view_creation option” [SQL Anywhere Server - Database Administration]
- “ALTER VIEW statement” on page 511

Example

The following example from a reload.sql script uses the sa_recompile_views system procedure to locate view definitions stored in the catalog that do not have column definitions and cause the column definitions to be created. Errors are ignored.

    CALL sa_recompile_views( 1 );

sa_refresh_materialized_views system procedure

Initializes all materialized views that are in an uninitialized state.

Syntax

    sa_refresh_materialized_views( [ ignore_errors ] )

Arguments

- **ignore_errors** Use this optional INTEGER parameter to specify whether to return errors during the recompilation. If you specify 0, an error is returned for each view for which column definition failed. If you specify 1, or any value other than 0, no errors are returned. The default is 0.

Remarks

A materialized view may be in an uninitialized state because it has just been created, has just been re-enabled, or the last attempt to initialize or refresh it failed due to an error. The sa_refresh_materialized_views system procedure scans the database for all such materialized views and attempts to initialize them. If the procedure encounters an error initializing a materialized view, it continues on attempting to process the remaining uninitialized views.

You can also use the REFRESH MATERIALIZED VIEW statement to initialize a materialized view.

Privileges

You must have the ALTER ANY MATERIALIZED VIEW system privilege.

Side effects

None

See also

- “REFRESH MATERIALIZED VIEW statement” on page 925
- “Manually refreshing a materialized view” [SQL Anywhere Server - SQL Usage]

Example

The following example uses the sa_refresh_materialized_views system procedure to initialize all materialized views that are in an uninitialized state. Errors are ignored.
CALL sa_refresh_materialized_views( 1 );

sa_refresh_text_indexes system procedure
Refreshes all text indexes defined as MANUAL REFRESH or AUTO REFRESH.

Syntax
sa_refresh_text_indexes( )

Remarks
The sa_refresh_text_indexes system procedure refreshes all text indexes defined as MANUAL REFRESH or AUTO REFRESH. It does not refresh text indexes defined as IMMEDIATE REFRESH (the default) because changes to those indexes are made when data is changed in the underlying table.

Privileges
You must have the CREATE ANY INDEX or ALTER ANY OBJECT system privilege.

Side effects
Automatic commit

See also
- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Text configuration object concepts and reference” [SQL Anywhere Server - SQL Usage]
- “DROP TEXT INDEX statement” on page 782
- “REFRESH TEXT INDEX statement” on page 928
- “TRUNCATE statement” on page 1019
- “SYSTEXTIDX system view” on page 1407
- “sa_text_index_stats system procedure” on page 1256
- “sa_text_index_vocab system procedure” on page 1258

Example
The following statement refreshes all MANUAL and AUTO REFRESH text indexes in the database:

    CALL sa_refresh_text_indexes();

sa_remove_tracing_data system procedure
Permanently deletes from the diagnostic tracing tables all records pertaining to the specified logging (tracing) session ID.

Syntax
sa_remove_tracing_data( log_session_id )
Arguments

- **log_session_id**  Use this UNSIGNED INTEGER parameter to specify the ID of the logging session for which to remove the data.

Remarks

If there are no records for the specified `log_session_id`, the procedure has no effect. The procedure has no return values.

Privileges

You must have the DIAGNOSTICS system role, and the MANAGE PROFILING system privilege.

Side effects

Causes a commit upon completion, even if no records were found for the specified `log_session_id`.

See also

- “Diagnostic tracing tables” on page 1069

Example

This example permanently deletes from the diagnostic tracing tables all records pertaining to the logging session ID 1.

```sql
CALL sa_remove_tracing_data( 1 );
```

### sa_report_deadlocks system procedure

Retrieves information about deadlocks from an internal buffer created by the database server.

Syntax

```sql
sa_report_deadlocks( )
```

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snapshotId</td>
<td>BIGINT</td>
<td>The deadlock instance (all rows pertaining to a particular deadlock have the same ID).</td>
</tr>
<tr>
<td>snapshotAt</td>
<td>TIMESTAMP</td>
<td>The time when the deadlock occurred.</td>
</tr>
<tr>
<td>waiter</td>
<td>INTEGER</td>
<td>The connection handle of the waiting connection.</td>
</tr>
<tr>
<td>who</td>
<td>VARCHAR(128)</td>
<td>The user ID associated with the connection that is waiting.</td>
</tr>
</tbody>
</table>
### Column name | Data type | Description
--- | --- | ---
what | LONG VARCHAR | The command being executed by the waiting connection. This information is only available if you have turned on capturing of the most recently-prepared SQL statement by specifying the -zl option on the database server command line or have turned this feature on using the sa_server_option system procedure.
object_id | UNSIGNED BIGINT | The object ID of the table containing the row.
record_id | BIGINT | The row ID of the associated row.
owner | INTEGER | The connection handle of the connection owning the lock being waited on.
is_victim | BIT | Identifies the rolled back transaction.
rollback_operation_count | UNSIGNED INTEGER | The number of uncommitted operations that may be lost if the transaction rolls back.

### Remarks
When the log_deadlocks option is set to On, the database server logs information about deadlocks in an internal buffer. You can view the information in the log using the sa_report_deadlocks system procedure.

### Privileges
You must have the MONITOR system privilege.

### Side effects
None

### See also
- “System events” [SQL Anywhere Server - Database Administration]
- “log_deadlocks option” [SQL Anywhere Server - Database Administration]
- “sa_server_option system procedure” on page 1222
- “How to determine who is blocked in a deadlock” [SQL Anywhere Server - SQL Usage]
- “-zl database server option” [SQL Anywhere Server - Database Administration]
- “sa_server_option system procedure” on page 1222

### Examples
You can execute the following query to identify deadlocks.

```sql
CALL sa_report_deadlocks();
```
sa_reserved_words system procedure

Returns a list of reserved words. Many of the keywords that appear in SQL statements are reserved words.

Syntax

```
sa_reserved_words()
```

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved_word</td>
<td>CHAR(128)</td>
<td>A reserved word.</td>
</tr>
</tbody>
</table>

Remarks

The procedure takes no parameters and returns one word per row. The list of reserved words is based on the version of the database server that executes the query, not the version of the software used to create the database file.

Privileges

None

Side effects

None

See also

- “Reserved words” on page 1

Example

The following statement returns a list of reserved words:

```
SELECT * FROM sa_reserved_words();
```

sa_reset_identity system procedure

Allows the next identity value to be set for a table. Use this procedure to change the AUTOINCREMENT value for the next row that will be inserted.

Syntax

```
sa_reset_identity(
    tbl_name
    [, owner_name
    [, new_identity ] ]
)
```
Arguments

- **tbl_name**  Use this CHAR(128) parameter to specify the table for which you want to reset the identity value. If the owner of the table is not specified, *tbl_name* must uniquely identify a table in the database.

- **owner_name**  Use this optional CHAR(128) parameter to specify the owner of the table for which you want to reset the identity value. The default is NULL. If *owner_name* is not specified, then use a named parameter value for the third argument. For example:

  ```sql
  CALL sa_reset_identity( 'Employees', new_identity=100 );
  ```

- **new_identity**  Use this optional BIGINT parameter to specify the value from which you want the autoincrementing to start. The default is NULL.

Remarks

The next identity value generated for a row inserted into the table is *new_identity* + 1.

No checking occurs to see whether *new_identity* + 1 conflicts with existing rows in the table. For example, if you specify *new_identity* as 100, the next row inserted gets an identity value of 101. However, if 101 already exists in the table, the row insertion fails.

The `sa_reset_identity` system procedure cannot be used on a table having no columns with a default of either AUTOINCREMENT or GLOBAL AUTOINCREMENT.

Privileges

You must be the owner of the table, or have the ALTER ANY TABLE system privilege.

Side effects

Causes a checkpoint to occur after the value has been updated.

See also

- “The AUTOINCREMENT default” [SQL Anywhere Server - SQL Usage]
- “The GLOBAL AUTOINCREMENT default” [SQL Anywhere Server - SQL Usage]

Example

The following statement resets the next identity value to 101:

```sql
CALL sa_reset_identity( 'Employees', 'GROUPO', 100 );
```

**sa_rowgenerator system procedure**

Returns a result set with rows between a specified start and end value.

**Syntax**

```
sa_rowgenerator(
  rstart
 [, rend ]
)
```
Arguments

- **rstart**  Use this optional INTEGER parameter to specify the starting value. The default value is 0.
- **rend**  Use this optional INTEGER parameter to specify the ending value that is greater than or equal to **rstart**. The default value is 100.
- **rstep**  Use this optional INTEGER parameter to specify the increment by which the sequence values are increased. The default value is 1.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_num</td>
<td>INTEGER</td>
<td>Sequence number.</td>
</tr>
</tbody>
</table>

Remarks

The sa_rowgenerator procedure can be used in the FROM clause of a query to generate a sequence of numbers. This procedure is an alternative to using the RowGenerator system table. You can use sa_rowgenerator for such tasks as:

- generating test data for a known number of rows in a result set.
- generating a result set with rows for values in every range. For example, you can generate a row for every day of the month, or you can generate ranges of zip codes.
- generating a query that has a specified number of rows in the result set. This may be useful for testing the performance of queries.

No rows are returned if you do not specify correct start and end values and a positive non-zero step value.

You can emulate the behavior of the RowGenerator table with the following statement:

```sql
SELECT row_num FROM sa_rowgenerator( 1, 255 );
```

Privileges

None

Side effects

None

See also

- “RowGenerator table (dbo)” on page 1083
- “Null value eliminated in aggregate function” [Error Messages]

Example

The following query returns a result set containing one row for each day of the current month.
SELECT DATEADD( day, row_num-1, 
    YMD( DATEPART( year, CURRENT DATE ),
    DATEPART( month, CURRENT DATE ), 1 ) ) 
AS day_of_month 
FROM sa_rowgenerator( 1, 31, 1 ) 
WHERE DATEPART( month, day_of_month ) = DATEPART( month, CURRENT DATE ) 
ORDER BY row_num;

The following query shows how many employees live in zip code ranges (0-9999), (10000-19999), ..., (90000-99999). Some of these ranges have no employees, which causes a warning.

The sa_rowgenerator procedure can be used to generate these ranges, even though no employees have a zip code in the range.

SELECT row_num AS r1, row_num+9999 AS r2, COUNT( PostalCode ) AS zips_in_range 
FROM sa_rowgenerator( 0, 99999, 10000 ) D LEFT JOIN Employees 
ON PostalCode BETWEEN r1 AND r2 
GROUP BY r1, r2 
ORDER BY 1;

The following example generates 10 rows of data and inserts them into the NewEmployees table:

INSERT INTO NewEmployees ( ID, Salary, Name )
SELECT row_num, CAST( RAND() * 1000 AS INTEGER ), 'Mary'
FROM sa_rowgenerator( 1, 10 );

The following example uses the sa_rowgenerator system procedure to create a view containing all integers. The value 2147483647 in this example represents the maximum signed integer that is supported.

CREATE VIEW Integers AS
SELECT row_num AS n
FROM sa_rowgenerator( 0, 2147483647, 1 );

This example uses the sa_rowgenerator system procedure to create a view containing dates from 0001-01-01 to 9999-12-31. The value 3652058 in this example represents the number of days between 0001-01-01 and 9999-12-31, the earliest and latest dates that are supported.

CREATE VIEW Dates AS
SELECT DATEADD( day, row_num, '0001-01-01' ) AS d
FROM sa_rowgenerator( 0, 3652058, 1 );

The following query returns all years between 1900 and 2058 that have 54 weeks.

SELECT DATEADD ( day, row_num, '1900-01-01' ) AS d, DATEPART ( week, d ) w
FROM sa_rowgenerator ( 0, 63919, 1 )
WHERE w = 54;

---

**sa_save_trace_data system procedure**

Saves tracing data to base tables.

**Syntax**

`sa_save_trace_data( )`
Remarks
While a tracing session is running, diagnostic data is stored in temporary versions of the diagnostic
tracing tables. When you stop a tracing session, you specify whether you want to permanently store the
tracing data in the base tables for diagnostic tracing. If you do not choose to save the data, you can still
save the data after the session is stopped by using the sa_save_trace_data system procedure.

The sa_save_trace_data system procedure returns an error if tracing is still in progress; you must stop
tracing to use this system procedure.

The sa_save_trace_data system procedure can be used even if the user specified WITHOUT SAVING
when stopping tracing. Also, the procedure must be called from the tracing database.

Privileges
You must have the MANAGE PROFILING system privilege.

Side effects
Automatic commit.

See also
● “Creating a diagnostic tracing session (Sybase Central)” [SQL Anywhere Server - SQL Usage]
● “Diagnostic tracing tables” on page 1069

Example
This example saves tracing data to the diagnostic tracing tables.

CALL sa_save_trace_data( );

**sa_send_udp system function**

Sends a UDP packet to the specified address.

Syntax
```
sa_send_udp(
    destAddress,
    destPort,
    msg
)
```

Arguments
- **destAddress** Use this CHAR(254) to specify either the host name or IP number.
- **destPort** Use this UNSIGNED SMALLINT parameter to specify the port number to use.
- **msg** Use this LONG BINARY parameter to specify the message to send to the specified address. If
  this value is a string, it must be enclosed in single quotes.
Returns
This function returns an INTEGER status code.

Remarks
This function sends a single UDP packet to the specified address. The function returns 0 if the message is sent successfully, and returns an error code if an error occurs. The error code is one of the following:

-1 if the message is too large to send over a UDP socket (as determined by the operating system) or if there is a problem with the destination address

the Winsock/Posix error code that is returned by the operating system

If the msg parameter contains binary data or is more complex than a string, you may want to use a BINARY variable. For example:

```
CREATE VARIABLE v LONG BINARY;
SET v='This is a UDP message';
SELECT sa_send_udp( '10.25.99.124', 1234, v );
DROP VARIABLE v;
```

This function can be used with MobiLink server-initiated synchronization to wake up the MobiLink Listener utility (dbsln.exe). If you use the sa_send_udp function as a way to notify the MobiLink Listener, you should append a 1 to your UDP packet. This number is a server-initiated synchronization protocol number. In future versions of MobiLink, new protocol versions may cause the MobiLink Listener to behave differently.

Privileges
You must have the MANAGE ANY WEB SERVICE system privilege.

Side effects
None

See also
- “Sending a push notification using the sa_send_udp system procedure” [MobiLink - Server-Initiated Synchronization]

Example
The following example sends the message "This is a test" to IP address 10.25.99.196 on port 2345:

```
SELECT sa_send_udp( '10.25.99.196', 2345, 'This is a test' );
```

```
SA_SERVER_MESSAGES system procedure
```
Allows you to return messages from the database server messages window as a result set.

Syntax
```
SA_SERVER_MESSAGES(
    {first_msg

```
Arguments

- **first_msg**  Use this optional UNSIGNED BIGINT parameter to specify the ID of the first or last message to be returned, depending on the sign of the `num_msgs` parameter. The default is NULL, which means that the search starts at the beginning of the list if `num_msgs` is NULL or non-negative; the search starts past the end of the list if `num_msgs` is negative.

- **num_msgs**  Use this optional BIGINT parameter to specify the number of messages to be returned. The sign indicates whether the request is for messages starting at `first_msg` or ending at `first_msg`. The default is NULL, which means that all messages starting at `first_msg` to the end of the list are returned.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg_id</td>
<td>UNSIGNED BIGINT</td>
<td>Unique message ID. Message IDs start at 0.</td>
</tr>
<tr>
<td>msg_text</td>
<td>LONG VARCHAR</td>
<td>Message text.</td>
</tr>
<tr>
<td>msg_time</td>
<td>TIMESTAMP</td>
<td>Time when the message was issued.</td>
</tr>
</tbody>
</table>
| msg_severity    | VARCHAR(255)       | Message severity. This column contains one of the values:
<p>|                 |                    | • <strong>INFO</strong> Informational message.                |
|                 |                    | • <strong>WARN</strong> Warning.                             |
|                 |                    | • <strong>ERR</strong> Error.                                 |</p>
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg_category</td>
<td>VARCHAR(255)</td>
<td>Message category. This column contains one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>STARTUP/SHUTDOWN</strong> Messages related to database server or database startup or shutdown.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>CHKPT</strong> Messages related to checkpoints.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>MSG</strong> Messages generated using the MESSAGE or PRINT statements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>DBA_MSG</strong> Messages generated using the MESSAGE statement that would have required the SERVER OPERATOR system privilege, such as messages sent to the event log.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>CONN</strong> Messages about database server connectivity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>OTHER</strong> All other types of messages.</td>
</tr>
<tr>
<td>msg_database</td>
<td>VARCHAR(255)</td>
<td>Database name associated with the message if it applies to one specific database. Otherwise, NULL.</td>
</tr>
</tbody>
</table>

**Remarks**

When new messages are sent to the console, old messages with the same category or severity are deleted if the number of messages exceeds the value of the MessageCategoryLimit property. As a result, there may be gaps in the result set, and two consecutive rows may not have consecutive message IDs.

The STARTUP/SHUTDOWN message category does not show shutdown messages for servers. Shutdown messages are shown only if multiple databases are running on a server and one or more are shut down.

**Privileges**

None

**Side effects**

None

**See also**

- “MessageCategoryLimit server property” [SQL Anywhere Server - Database Administration]

**Example**

The following command requests 100 messages starting at the message whose ID is 3:

```sql
CALL sa_server_messages( 3, 100 );
```
The following command requests 500 messages up to, and including, message 4032:

```sql
CALL sa_server_messages( 4032, -500 );
```

The following commands request all messages starting with message 3:

```sql
CALL sa_server_messages( 3, NULL );
CALL sa_server_messages( 3 );
```

The following command requests the first 100 messages in the list:

```sql
CALL sa_server_messages( NULL, 100 );
```

The following command requests the last 100 messages in the list:

```sql
CALL sa_server_messages( NULL, -100 );
```

The following commands request all the messages in the list:

```sql
CALL sa_server_messages( NULL, NULL );
CALL sa_server_messages( );
```

### sa_server_option system procedure

Overrides a server option while the server is running.

**Syntax**

```sql
sa_server_option(
    opt,
    val
)
```

**Arguments**

- **opt** Use this CHAR(128) parameter to specify a server option name.
- **val** Use this CHAR(128) parameter to specify the new value for the server option.

**Remarks**

Database administrators can use this procedure to override some database server options temporarily, without restarting the database server.

The option values that are changed using this procedure are reset to their default values when the database server shuts down. To change an option value every time you start the database server, specify the corresponding database server option when the database server is started (if one exists).

The following option settings can be changed. Default values are shown in bold:
<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoMultiProgrammingLevel</td>
<td>YES, NO</td>
<td>When set to YES, the database server automatically adjusts its multiprogramming level, which controls the maximum number of tasks that can be active at a time. If you choose to control the multiprogramming level manually by setting this option to NO, you can still set the initial, minimum, and maximum values for the multiprogramming level. See “-gna database server option” [SQL Anywhere Server - Database Administration] and “Database server configuration of the multiprogramming level” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>AutoMultiProgrammingLevelStatistics</td>
<td>YES, NO</td>
<td>When set to YES, statistics for automatic multiprogramming level adjustments appear in the database server message log. See “-gns database server option” [SQL Anywhere Server - Database Administration] and “AutoMultiProgrammingLevelStatistics server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>CacheSizingStatistics</td>
<td>YES, NO</td>
<td>When set to YES, display cache information in the database server messages window whenever the cache size changes. See “-cs database server option” [SQL Anywhere Server - Database Administration] and “CacheSizingStatistics server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>CollectStatistics</td>
<td>YES, NO</td>
<td>When set to YES, the database server collects Performance Monitor statistics. See “-k database server option” [SQL Anywhere Server - Database Administration] and “CollectStatistics server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ConnsDisabled</td>
<td>YES, NO</td>
<td>When set to YES, no other connections are allowed to any databases on the database server. See “ConnsDisabled server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>ConnsDisabled-ForDB</td>
<td>YES, NO</td>
<td>When set to YES, no other connections are allowed to the current database.</td>
</tr>
<tr>
<td>Option name</td>
<td>Values</td>
<td>Additional information</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Console-LogFile</td>
<td>filename</td>
<td>The name of the file used to record database server message log information. Specifying an empty string stops logging to the file. Double any backslash characters in the path because this value is a SQL string. See “--o database server option” [SQL Anywhere Server - Database Administration] and “ConsoleLogFile server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Console-LogMax-Size</td>
<td>file-size, in bytes</td>
<td>The maximum size, in bytes, of the file used to record database server message log information. When the database server message log file reaches the size specified by either this property or the -on server option, the file is renamed with the extension .old appended (replacing an existing file with the same name if one exists). The database server message log file is then restarted. See “--on database server option” [SQL Anywhere Server - Database Administration] and “ConsoleLogMaxSize server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Current-Multi-ProgrammingLevel</td>
<td>Integer. Default is 20.</td>
<td>Sets the multiprogramming level of the database server.</td>
</tr>
<tr>
<td>DatabaseCleaner</td>
<td>ON, OFF</td>
<td>Do not change the setting of this option except on the recommendation of Technical Support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See “DatabaseCleaner database property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Option name</td>
<td>Values</td>
<td>Additional information</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| DeadlockLogging             | ON, OFF, RESET, CLEAR | Controls deadlock logging. The value deadlock_logging is also supported. Deadlock logging options can also be configured in the **Database Properties** window in Sybase Central. The following values are supported:  
  - **ON** Enables deadlock logging.  
  - **OFF** Disables deadlock logging and leaves the deadlock data available for viewing.  
  - **RESET** Clears the logged deadlock data, if any exists, and then enables deadlock logging.  
  - **CLEAR** Clears the logged deadlock data, if any exists, and then disables deadlock logging.  
  Once deadlock logging is enabled, you can use the sa_report_deadlocks system procedure to retrieve deadlock information from the database.  
  See “log_deadlocks option” [SQL Anywhere Server - Database Administration]. |
| DebuggingInformation        | YES, NO         | Displays diagnostic messages and other messages for troubleshooting purposes. The messages appear in the database server messages window.  
  See “-z database server option” [SQL Anywhere Server - Database Administration] and “DebuggingInformation server property” [SQL Anywhere Server - Database Administration]. |
| DiskSandbox                 | ON, OFF         | Sets the default disk sandbox settings for all databases started on the database server that do not have explicit disk sandbox settings. Changing the disk sandbox settings by using the sa_server_option system procedure does not affect databases already running on the database server. To use the sa_server_option system procedure to change disk sandbox settings, you must provide the secure feature key for the manage_disk_sandbox secure feature.  
  See “-sbx database server option” [SQL Anywhere Server - Database Administration]. |
| DropBadStatistics           | YES, NO         | Allows automatic statistics management to drop statistics that return bad estimates from the database.                                                                                                               |
| DropUnusedStatistics        | YES, NO         | Allows automatic statistics management to drop statistics that have not been used for 90 consecutive days from the database.  
  See “-sbx database server option” [SQL Anywhere Server - Database Administration]. |
<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>IdleTime-out</td>
<td>Integer, in minutes. The default is <strong>240</strong>.</td>
<td>Disconnects TCP/IP connections that have not submitted a request for the specified number of minutes. This prevents inactive connections from holding locks indefinitely. See “-ti database server option” [SQL Anywhere Server - Database Administration] and “IdleTimeout server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>IPAddressMonitorPeriod</td>
<td>Integer, in seconds. The default is <strong>120</strong> for portable devices, <strong>0</strong> otherwise.</td>
<td>Sets the time to check for new IP addresses in seconds. The minimum value is 10 and the default is 0. For portable devices, the default value is 120 seconds. See “-xm database server option” [SQL Anywhere Server - Database Administration] and “IPAddressMonitorPeriod server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>LivenessTimeout</td>
<td>Integer, in seconds. The default is <strong>120</strong>.</td>
<td>A liveness packet is sent periodically across a client/server TCP/IP network to confirm that a connection is intact. If the network server runs for a LivenessTimeout period without detecting a liveness packet, the communication is severed. See “-tl database server option” [SQL Anywhere Server - Database Administration] and “LivenessTimeout server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>MaxMultiProgrammingLevel</td>
<td>Integer. The default is four times the value for CurrentMultiProgrammingLevel</td>
<td>Sets the maximum database server multiprogramming level. See “-gnh database server option” [SQL Anywhere Server - Database Administration] and “Database server configuration of the multiprogramming level” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>MessageCategoryLimit</td>
<td>Integer. The default is <strong>400</strong>.</td>
<td>Sets the minimum number of messages of each severity and category that can be retrieved using the sa_server_messages system procedure. See “MessageCategoryLimit server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Option name</td>
<td>Values</td>
<td>Additional information</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MinMultiprogrammingLevel</td>
<td>Integer.</td>
<td>Sets the minimum database server multiprogramming level. The default is the minimum of the value of the -gtc server option and the number of logical CPUs on the computer. See “-gtc database server option” [SQL Anywhere Server - Database Administration] and “Database server configuration of the multiprogramming level” [SQL Anywhere Server - Database Administration].</td>
</tr>
</tbody>
</table>
| OptionWatchAction                   | MESSAGE,   | Specifies the action that the database server takes when an attempt is made to set an option in the list. The supported values are MESSAGE and ERROR. When OptionWatchAction is set to MESSAGE, and an option specified by OptionWatchList is set, a message appears in the database server messages window indicating that the option being set is on the options watch list. When OptionWatchAction is set to ERROR, an error is returned indicating that the option cannot be set because it is on the options watch list. You can view the current setting for this property by executing the following query:  

```sql
SELECT DB_PROPERTY( 'OptionWatchAction' );
```

See “Monitoring option settings” [SQL Anywhere Server - Database Administration] and “OptionWatchList database property” [SQL Anywhere Server - Database Administration]. | ERROR       |
<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
</table>
| Option-Watch-List | Comma-separated list of database options. | Specifies a comma-separated list of database options that you want to be notified about, or have the database server return an error for, when they are set. The string length is limited to 128 bytes. By default, it is an empty string. For example, the following command adds the automatic_timestamp, float_as_double, and tsql_hex_constant option to the list of options being watched:  

```sql
CALL sa_server_option( 'OptionWatchList','automatic_timestamp, float_as_double,tsql_hex_constant' );
```

You can view the current setting for this property by executing the following query:

```sql
SELECT DB_PROPERTY( 'OptionWatchList' );
```

See “Monitoring option settings” [SQL Anywhere Server - Database Administration] and “OptionWatchAction database property” [SQL Anywhere Server - Database Administration]. |
| ProcedureProfiling | YES, NO, RESET, CLEAR | See “ProcedureProfiling database property” [SQL Anywhere Server - Database Administration]. |
| Profile-Filter-Conn | connection-id | Instructs the database server to capture profiling information for a specific connection ID, without preventing other connections from using the database. When connection filtering is enabled, the value returned for `SELECT PROPERTY( 'ProfileFilterConn' )` is the connection ID of the connection being monitored. If no ID has been specified, or if connection filtering is disabled, the value returned is -1.

See “ProfileFilterConn server property” [SQL Anywhere Server - Database Administration]. |
<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
</table>
| ProcessorAffinity   | Comma-delimited list of       | Instructs the database server which logical processors to use on Windows or Linux. Specify a comma-delimited list of processor numbers and/or ranges. If the lower endpoint of a range is omitted, then it is assumed to be zero. If the upper endpoint of a range is omitted, then it is assumed to be the highest CPU known to the operating system. The in_use column returned by the sa_cpu_topology system procedure contains the current processor affinity of the database server, and the in_use column indicates whether the database server is using a processor. Alternatively, you can query the value of the ProcessorAffinity database server property. The database server might not use all of the specified logical processors in the following cases:  
  - If one or more of the specified logical processors does not exist, or is offline.  
  - If the license does not allow it.  
  If you specify an invalid processor ID, sa_server_option returns an error.  
  See “-gta database server option” [SQL Anywhere Server - Database Administration] and “sa_cpu_topology system procedure” on page 1116. |
<p>|                     | processor numbers and/or      |                                                                                                                                                                                                                                                                                                                                                       |
|                     | ranges. The default is that   |                                                                                                                                                                                                                                                                                                                                                       |
|                     | all processors are used or    |                                                                                                                                                                                                                                                                                                                                                       |
|                     | the setting of the -gta       |                                                                                                                                                                                                                                                                                                                                                       |
|                     | option.                       |                                                                                                                                                                                                                                                                                                                                                       |
| ProfileFilterUser   | user-id                       | Instructs the database server to capture profiling information for a specific user ID.                                                                                                                                                                                                                                                                  |
|                     |                               | See “ProfileFilterUser server property” [SQL Anywhere Server - Database Administration].                                                                                                                                                                                                                                                              |
| QuittingTime        | Valid date and time           | Instructs the database server to shut down at the specified time.                                                                                                                                                                                                                                                                                       |
|                     |                               | See “-tq database server option” [SQL Anywhere Server - Database Administration] and “QuittingTime server property” [SQL Anywhere Server - Database Administration].                                                                                                                                                                                  |</p>
<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>RememberLastPlan</td>
<td>YES, NO</td>
<td>Instructs the database server to capture the long text plan of the last query executed on the connection. This setting is also controlled by the -zp server option. When RememberLastPlan is turned on, obtain the textual representation of the plan of the last query executed on the connection by querying the value of the LastPlanText connection property: SELECT CONNECTION_PROPERTY( 'LastPlanText' ); See “-zp database server option” [SQL Anywhere Server - Database Administration] and “RememberLastPlan server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Option name</td>
<td>Values</td>
<td>Additional information</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| RememberLastStatement | YES, NO    | Instructs the database server to capture the most recently prepared SQL statement for each database running on the server. For stored procedure calls, only the outermost procedure call appears, not the statements within the procedure. When RememberLastStatement is turned on, you can obtain the current value of the LastStatement for a connection by querying the value of the LastStatement connection property:  
`SELECT CONNECTION_PROPERTY( 'LastStatement' );`  
When client statement caching is enabled, and a cached statement is reused, this property returns an empty string. When RememberLastStatement is turned on, the following statement returns the most recently-prepared statement for the specified connection:  
`SELECT CONNECTION_PROPERTY( 'LastStatement', connection-id );`  
The sa_conn_activity system procedure returns this same information for all connections. **Caution**  
When -zl is specified, or when the RememberLastStatement server setting is turned on, any user can call the sa_conn_activity system procedure or obtain the value of the LastStatement connection property to find out the most recently-prepared SQL statement for any other user. Use this option with caution and turn it off when it is not required.  
See “-zl database server option” [SQL Anywhere Server - Database Administration] and “RememberLastStatement server property” [SQL Anywhere Server - Database Administration].
### System procedures

<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
</table>
| RequestFilterConn | connection-id, -1 | Filter the request logging information so that only information for a particular connection is logged. This filtering can reduce the size of the request log file when monitoring a database server with many active connections or multiple databases. You can obtain the connection ID by executing the following:  
```sql
CALL sa_conn_info();
```
To log a specific connection once you have obtained the connection ID, execute the following statement:  
```sql
CALL sa_server_option('RequestFilterConn', connection-id);
```
Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:  
```sql
CALL sa_server_option('RequestFilterConn', -1);
```
See “RequestFilterConn server property” [SQL Anywhere Server - Database Administration]. |
| RequestFilterDB   | database-id, -1  | Filter the request logging information so that only information for a particular database is logged. This can help reduce the size of the request log file when monitoring a server with multiple databases. You can obtain the database ID by executing the following statement when you are connected to the desired database:  
```sql
SELECT CONNECTION_PROPERTY('DBNumber');
```
To log only information for a particular database, execute the following statement:  
```sql
CALL sa_server_option('RequestFilterDB', database-id);
```
Filtering remains in effect until it is explicitly reset, or until the database server is shut down. To reset filtering, use the following statement:  
```sql
CALL sa_server_option('RequestFilterDB', -1);
```
See “RequestFilterDB server property” [SQL Anywhere Server - Database Administration]. |
<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestLogFile</td>
<td><em>filename</em></td>
<td>The name of the file used to record request information. Specifying an empty string stops logging to the request log file. If request logging is enabled, but the request log file was not specified or has been set to an empty string, the server logs requests to the database server messages window. Double any backslash characters in the path because this value is a SQL string. When client statement caching is enabled, set the max_client_statements_cached option to 0 to disable client statement caching while the request log is captured, if the log will be analyzed using the tracetime.pl Perl script. See “-zo database server option” [SQL Anywhere Server - Database Administration] and “RequestLogFile server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Option name</td>
<td>Values</td>
<td>Additional information</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| RequestLogging | SQL, HOSTVARS, PLAN, PROCEDURES, TRIGGERS, OTHER, BLOCKS, REPLACE, ALL, YES, NONE, NO | This call turns on logging of individual SQL statements sent to the database server for use in troubleshooting with the database server -zr and -zo options. Values can be combinations of the following, separated by either a plus sign (+), or a comma:  
- **PLAN** enables logging of execution plans (short form). If logging of procedures (PROCEDURES) is enabled, execution plans for procedures are also recorded.  
- **HOSTVARS** enables logging of host variable values. If you specify HOSTVARS, the information listed for SQL is also logged.  
- **PROCEDURES** enables logging of statements executed from within procedures.  
- **TRIGGERS** enables logging of statements executed from within triggers.  
- **OTHER** enables logging of additional request types not included by SQL, such as FETCH and PREFETCH. However, if you specify OTHER but do not specify SQL, it is the equivalent of specifying SQL+OTHER. Including OTHER can cause the log file to grow rapidly and could negatively impact server performance.  
- **BLOCKS** enables logging of details showing when a connection is blocked and unblocked on another connection.  
- **REPLACE** at the start of logging, the existing request log is replaced with a new (empty) one of the same name. Otherwise, the existing request log is opened and new entries are appended to the end of the file.  
- **ALL** logs all supported information. This value is equivalent to specifying SQL+PLAN+HOSTVARS+PROCEDURES+TRIGGERS+OTHER+BLOCKS. This setting can cause the log file to grow rapidly and could negatively impact server performance.  
- **NO or NONE** turns off logging to the request log.  
You can view the current setting for this property by executing the following query:  
```
SELECT PROPERTY( 'RequestLogging' );
```
For more information, see “List of database server properties” [SQL Anywhere Server - Database Administration].
<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestLogMaxSize</td>
<td>file-size, in bytes</td>
<td>The maximum size of the file used to record request logging information, in bytes. If you specify 0, then there is no maximum size for the request logging file, and the file is never renamed. This value is the default. When the request log file reaches the size specified by either the sa_server_option system procedure or the -zs server option, the file is renamed with the extension .old appended (replacing an existing file with the same name if one exists). The request log file is then restarted. See “-zs database server option” [SQL Anywhere Server - Database Administration] and “RequestLogMaxSize server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>RequestLogNumFiles</td>
<td>Integer</td>
<td>The number of request log file copies to retain. If request logging is enabled over a long period, the request log file can become large. The -zn option allows you to specify the number of request log file copies to retain. See “-zn database server option” [SQL Anywhere Server - Database Administration] and “RequestLogNumFiles server property” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>Option name</td>
<td>Values</td>
<td>Additional information</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>------------------------</td>
</tr>
</tbody>
</table>
| RequestTiming | YES, NO | Instructs the database server to maintain timing information for each new connection. This feature is turned off by default. When it is turned on, the database server maintains cumulative timers for all new connections that indicate how much time the connection spent in the server in each of several states. The change is only effective for new connections, and lasts for the duration each connection. You can use the `sa_performance_diagnostics` system procedure to obtain a summary of this timing information, or you can retrieve individual values by inspecting the following connection properties:  
- ReqCountUnscheduled  
- ReqTimeUnscheduled  
- ReqCountActive  
- ReqTimeActive  
- ReqCountBlockIO  
- ReqTimeBlockIO  
- ReqCountBlockLock  
- ReqTimeBlockLock  
- ReqCountBlockContention  
- ReqTimeBlockContention  
When the RequestTiming server property is on, there is a small overhead for each request to maintain the additional counters. See “-zt database server option” [SQL Anywhere Server - Database Administration] and “RequestTiming server property” [SQL Anywhere Server - Database Administration]. |
<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
</table>
| Secure-Features   | feature-list | Allows you to manage secure features for a database server that is already running. The feature-list is a comma-separated list of feature names or feature sets. By adding a feature to the list, you limit its availability. To remove items from the list of secure features, specify a minus sign (-) before the secure feature name.  

To call sa_server_option('SecureFeatures',...), the connection must have the ManageFeatures secure feature enabled on the connection. The -sf key (the system secure feature key) enables ManageFeatures, as well as all of the other features. So if you used the system secure feature key, then changing the set of SecureFeatures will not have any effect on the connection. But if you used another key (for example a key that had been created using the create_secure_feature_key system procedure) then your connection may be immediately affected by the change, depending on what other features are included in the key.  

For the list of secure feature features, see “-sf database server option” [SQL Anywhere Server - Database Administration]  
Any changes you make to allow or prevent access to features take effect immediately for the database server. The connection that executes the sa_server_option system procedure may or may not be affected, depending on the secure feature key the connection is using and whether or not it allows the connection access to the specified features.  
For example, to secure two features, use the following syntax:  
CALL sa_server_option('SecureFeatures', 'CONSOLE_LOG,WEBCLIENT_LOG');  
After executing this statement, the list of secure features is set according to what has been changed.  
See “-sf database server option” [SQL Anywhere Server - Database Administration] and “Creating secure feature keys” [SQL Anywhere Server - Database Administration]. |
| StatisticsCleaner | ON, OFF  | The statistics cleaner fixes statistics that give bad estimates by performing scans on tables. By default the statistics cleaner runs in the background and has a minimal impact on performance.  
Turning off the statistics cleaner does not disable the statistic governor, but when the statistics cleaner is turned off, statistics are only created or fixed when a query is run. |
<table>
<thead>
<tr>
<th>Option name</th>
<th>Values</th>
<th>Additional information</th>
</tr>
</thead>
</table>
| Web-ClientLogFile     | filename       | The name of the web service client log file. The web service client log file is truncated each time you use the -zoc server option or the WebClientLogFile property to set or reset the file name. Double any backslash characters in the path because this value is a string.  
  See “-zoc database server option” [SQL Anywhere Server - Database Administration] and “WebClientLogFile server property” [SQL Anywhere Server - Database Administration]. |
| Web-ClientLogging     | ON, OFF        | This option enables and disables logging of web service clients. The information that is logged includes HTTP requests and response data. Specify ON to start logging to the web service client log file, and specify OFF to stop logging to the file.  
  See “-zoc database server option” [SQL Anywhere Server - Database Administration] and “WebClientLogging server property” [SQL Anywhere Server - Database Administration]. |

**Privileges**

You must have the MANAGE PROFILING system privilege to use the following options, which are related to application profiling or request logging:

- ProcedureProfiling
- ProfileFilterConn
- ProfileFilterUser
- RequestFilterConn
- RequestFilterDB
- RequestLogFile
- RequestLogging
- RequestLogMaxSize
- RequestLogNumFiles

For all other options, your must have the SERVER OPERATOR system privilege.

**Side effects**

None
See also

- “Procedure profiling using system procedures” [SQL Anywhere Server - SQL Usage]
- “Tip: Update column statistics” [SQL Anywhere Server - SQL Usage]
- “sa_performance_diagnostics system procedure” on page 1196
- “List of connection properties” [SQL Anywhere Server - Database Administration]
- “log_deadlocks option” [SQL Anywhere Server - Database Administration]
- “-sk database server option” [SQL Anywhere Server - Database Administration]
- “-sf database server option” [SQL Anywhere Server - Database Administration]
- “sa_db_option system procedure” on page 1120

Example

The following statement causes cache information to be displayed in the database server messages window whenever the cache size changes:

```sql
CALL sa_server_option( 'CacheSizingStatistics', 'YES' );
```

The following statement disallows new connections to the current database:

```sql
CALL sa_server_option( 'ConnsDisabledForDB', 'YES' );
```

The following statement enables logging of all SQL statements, procedure calls, plans, blocking and unblocking events, and starts a new request log:

```sql
CALL sa_server_option( 'RequestLogging', 'SQL+PROCEDURES+BLOCKS+PLAN+REPLACE' );
```

### sa_set_http_header system procedure

Permits a web service to set an HTTP response header.

**Syntax**

```sql
sa_set_http_header(
    fldname
    , val
)
```

**Arguments**

- `fldname` Use this CHAR(128) parameter to specify a string containing the name of one of the HTTP header fields.

- `val` Use this LONG VARCHAR parameter to specify the value to which the named parameter should be set. Setting a response header to NULL, effectively removes it.

**Remarks**

Setting the special header field @HttpStatus sets the status code returned with the request. The status code is also known as the response code. For example, the following script sets the status code to 404 Not Found:

```sql
CALL sa_set_http_header( '@HttpStatus', '404' );
```
You can create a user-defined status message by specifying a three digit status code with an optional colon-delimited text message. For example, the following script outputs a status code with the message "999 User Code":

```sql
CALL sa_set_http_header( '@HttpStatus', '999:User Code' );
```

**Note**
A user defined status text message is not translated into a database character-set when logged using the LogOptions protocol option.

The body of the error message is inserted automatically. Only valid HTTP error codes can be used.
Setting the status to an invalid code causes a SQL error.

The `sa_set_http_header` procedure always overwrites the existing header value of the header field when called.

Response headers generated automatically by the database server can be removed. For example, the following command removes the Expires response header:

```sql
CALL sa_set_http_header( 'Expires', NULL );
```

**Privileges**
None

**Side effects**
None

**See also**
- “NEXT_HTTP_RESPONSE_HEADER function [Web service]” on page 308
- “HTTP_RESPONSE_HEADER function [Web service]” on page 268
- “Web services functions” on page 151
- “Web services system procedures” on page 1085
- “LogOptions (LOPT) protocol option” [SQL Anywhere Server - Database Administration]

**Example**
The following example sets the Content-Type header field to text/html.

```sql
CALL sa_set_http_header( 'Content-Type', 'text/html' );
```

**sa_set_http_option system procedure**
Permits a web service to set an HTTP option for process control.

**Syntax**
```sql
sa_set_http_option(
  optname
, val
)
```
Arguments

- **optname**  Use this CHAR(128) parameter to specify a string containing the name of one of the HTTP options.

The supported options are:

- **CharsetConversion**  Use this option to control whether the result set is to be automatically converted from the character set encoding of the database to the character set encoding of the client. The only permitted values are ON and OFF. The default value is ON.

- **AcceptCharset**  Use this option to specify the web server's preferences for a response character set encoding. One or more character set encodings may be specified in order of preference. The syntax for this option conforms to the syntax used for the HTTP Accept-Charset request-header field specification in RFC2616 Hypertext Transfer Protocol.

An HTTP client such as a web browser may provide an Accept-Charset request header which specifies a list of character set encodings ordered by preference. Optionally, each encoding may be given an associated quality value \((q=qvalue)\) which represents the client's preference for that encoding. By default, the quality value is 1 \((q=1)\). Here is an example:

```
Accept-Charset: iso-8859-5, utf-8;q=0.8
```

A plus sign (+) in the AcceptCharset HTTP option value may be used as a shortcut to represent the current database character set encoding. The plus sign also indicates that the database character set encoding should take precedence if the client also specifies the encoding in its list, regardless of the quality value assigned by the client.

An asterisk (*) in the AcceptCharset HTTP option may be used to indicate that the web service should use a character set encoding preferred by the client, as long as it is also supported by the server, when client and server do not have an intersecting list.

When sending the response, the first character set encoding preferred by both client and web service is used. The client's order of preference takes precedence. If no mutual encoding preference exists, then the web service's most preferred encoding is used, unless an asterisk (*) appears in the web service list in which case the client's most preferred encoding is used.

If the AcceptCharset HTTP option is not used, the most preferred character set encoding specified by the client and supported by the server is used. If none of the encodings specified by the client are supported (or the client does not send an Accept-Charset request header) then the database character set encoding is used.

If a client does not send an Accept-Charset header then one of the following actions are taken:

- If the AcceptCharset HTTP option has not been specified then the web server will use the database character set encoding.

- If the AcceptCharset HTTP option has been specified then the web server will use its most preferred character set encoding.

If a client does send an Accept-Charset header then one of the following actions are taken:
If the AcceptCharset HTTP option has not been specified then the web server will attempt to use one of the client's preferred character set encodings, starting with the most preferred encoding. If the web server does not support any of the client's preferred encodings, it will use the database character set encoding.

If the AcceptCharset HTTP option has been specified then the web server will attempt to use the first preferred character set encoding common to both lists, starting with the client's most preferred encoding. For example, if the client sends an Accept-Charset header listing, in order of preference, encodings iso-a, iso-b, and iso-c and the web server prefers iso-b, then iso-a, and finally iso-c, then iso-a will be selected.

Web client: iso-a, iso-b, iso-c
Web server: iso-b, iso-a, iso-c

If the intersection of the two lists is empty, then the web server's first preferred character set is used. From the following example, encoding iso-d will be used.

Web client: iso-a, iso-b, iso-c
Web server: iso-d, iso-e, iso-f

If an asterisk ('*') was included in the AcceptCharset HTTP option, then emphasis would be placed on the client's choice of encodings, resulting in iso-a being used. Essentially, the use of an asterisk guarantees that the intersection of the two lists will not be empty.

The ideal situation occurs when both client and web service use the database character set encoding since this eliminates the need for character set translation and improves the response time of the web server.

If the CharsetConversion option has been set to OFF, then AcceptCharset processing is not performed.

- **SessionID** Use this option to create, delete or rename an HTTP session. The database connection is persisted when a web service sets this option to create an HTTP session but sessions are not persisted across server restarts. If already within a session context, this call will rename the session to the new session ID. When called with a NULL value, the session will be deleted when the web service terminates.

  The generated session keys are limited to 128 characters in length and unique across databases if multiple databases are loaded.

- **SessionTimeout** Use this option to specify the amount of time, in minutes, that the HTTP session persists during inactivity. This time-out period is reset whenever an HTTP request uses the given session. The session is automatically deleted when the SessionTimeout is exceeded.

- **val** Use this LONG VARCHAR parameter to specify the value to which the named option should be set.

**Remarks**

Use this procedure within statements or procedures that handle web services to set options.

When sa_set_http_option is called from within a procedure invoked through a web service, and either the option or option value is invalid, an error is returned.
Privileges

None

Side effects

None

See also

- “Character set conversion considerations” [SQL Anywhere Server - Programming]
- “HTTP session management on an HTTP server” [SQL Anywhere Server - Programming]
- “List of connection properties” [SQL Anywhere Server - Database Administration]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Examples

The following example illustrates the use of sa_set_http_option to indicate the web service's preference for database character set encoding. The UTF-8 encoding is specified as a second choice. The asterisk (*) indicates that the web service is willing to use the character set encoding most preferred by the client, provided that it is supported by the web server.

```
CALL sa_set_http_option( 'AcceptCharset', '+,UTF-8,*');
```

The following example illustrates the use of sa_set_http_option to correctly identify the character encoding in use by the web service. In this example, the web server is connected to a 1251CYR database and is prepared to serve HTML documents containing the Cyrillic alphabet to any web browser.

```
CREATE OR REPLACE PROCEDURE cyrillic_html()
RESULT (html_doc XML)
BEGIN
    DECLARE pos INT;
    DECLARE charset VARCHAR(30);
    CALL sa_set_http_option( 'AcceptCharset', 'iso-8859-5, utf-8' );
    SET charset = CONNECTION_PROPERTY( 'CharSet' );
    -- Change any IANA labels like ISO_8859-5:1988
    -- to ISO_8859-5 for Firefox.
    SET pos = LOCATE( charset, ':' );
    IF pos > 0 THEN
        SET charset = LEFT( charset, pos - 1 );
    END IF;
    CALL sa_set_http_header( 'Content-Type', 'text/html; charset=' || charset );
    SELECT '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"' ||
        XMLCONCAT(
        XML要素('HTML',
            XML要素('HEAD',
                XML要素('TITLE', 'Cyrillic characters')
            ),
            XML要素('BODY',
                XML要素('H1', 'First 5 lowercase Russian letters'),
                XML要素('P', UNISTR('\u0430\u0431\u0432\u0433\u0434'))
            )
        )
    )
END;
```

CREATE SERVICE cyrillic
    TYPE 'RAW'
To illustrate the process of establishing the correct character set encoding to use, consider the following Accept-Charset header delivered by a web browser such as Firefox to the web service. It indicates that the browser prefers ISO-8859-1 and UTF-8 encodings but is willing to accept others.

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

The web service will not accept the ISO-8859-1 character set encoding since the web page to be transmitted contains Cyrillic characters. The web service prefers ISO-8859-5 or UTF-8 encodings as indicated by the call to sa_set_http_option. In this example, the UTF-8 encoding will be chosen since it is agreeable to both parties. The database connection property CharSet indicates which encoding has been selected by the web service. The sa_set_http_header procedure is used to indicate the HTML document's encoding to the web browser.

```
Content-Type: text/html; charset=UTF-8
```

If the web browser does not specify an Accept-Charset, then the web service defaults to its first preference, ISO-8859-5. The sa_set_http_header procedure is used to indicate the HTML document's encoding.

```
Content-Type: text/html; charset=ISO_8859-5
```

The following example sets a unique HTTP session identifier:

```
BEGIN
  DECLARE sessionid VARCHAR(30);
  DECLARE tm TIMESTAMP;
  SET tm = NOW(*);
  SET sessionid = 'MySessions_' || CONVERT( VARCHAR, SECONDS(tm)*1000 + DATEPART(millisecond,tm));
  SELECT sessionid;
  CALL sa_set_http_option('SessionID', sessionid);
END;
```

The following example sets the time-out for an HTTP session to 5 minutes:

```
CALL sa_set_http_option('SessionTimeout', '5');
```

### sa_set_soap_header system procedure

Permits the setting of SOAP headers for SOAP responses. This procedure is used within stored procedures called from SOAP web services.

**Syntax**

```
sa_set_soap_header(
  field_name  
  , val
)
```
Arguments

- **fldname**  Use this CHAR(128) parameter to specify the header key, a unique string used to reference the given header entry (it need not be identical to the localname of the *val*).

- **val**  Use this LONG VARCHAR parameter to specify the raw XML of a top level header entry and its children within the scope of a SOAP Header element.

Remarks

All SOAP header entries set with this procedure are serialized within the SOAP Header element when the SOAP response message is sent. A *val* of NULL is not serialized. If no header entries exist for a SOAP response, then an enclosing Header element, within the SOAP envelope, is not created.

Privileges

None

Side effects

None

See also

- “Tutorial: Using SQL Anywhere to access a SOAP/DISH service” [SQL Anywhere Server - Programming]
- “Web services functions” on page 151
- “Web services system procedures” on page 1085

Example

The following example sets the SOAP header welcome to Hello:

```sql
CALL sa_set_soap_header( 'welcome', '<welcome>Hello</welcome>' )
```

**sa_set_tracing_level system procedure**

Initializes the level of tracing information to be stored in the diagnostic tracing tables.

Syntax

```sql
sa_set_tracing_level(
    level
    [, specified_scope
    [, specified_name
    [, do_commit ] ] ] ]
)
```

Arguments

- **level**  Use this INTEGER parameter to specify the level of diagnostic tracing to perform. Possible values include:

  - 0  Do not generate any tracing data. This level keeps the tracing session open, but does not send any tracing data to the diagnostic tracing tables.
○ 1  Sets a basic level of tracing.
○ 2  Sets a medium level of tracing.
○ 3  Sets a high level of tracing.

- **specified_scope**  Use this optional LONG VARCHAR parameter to specify the tracing scope; for example, USER, DATABASE, CONNECTION_NAME, TRIGGER, and so on. The default is NULL.

- **specified_name**  Use this optional LONG VARCHAR parameter to specify the identifier for the object indicated in specified_scope. The default is NULL.

- **do_commit**  Use this optional TINYINT parameter to specify whether to commit, automatically, rows inserted by this procedure. The default is 1. Specify 1 to commit the rows automatically (recommended), and 0 to not commit them automatically.

**Remarks**

This procedure replaces the rows in the sa_diagnostic_tracing_level table, changing the tracing level and scope to the settings specified when the procedure is called.

Setting the level 0 does not stop the tracing session. Instead, the tracing session remains attached to the tracing database, but no tracing data is sent. The tracing session is still active when the level is 0.

This system procedure must be called from the database being profiled.

**Privileges**

You must have the DIAGNOSTICS system role, and the MANAGE PROFILING system privilege.

**Side effects**

None

**See also**

- “Customized diagnostic tracing levels” [SQL Anywhere Server - SQL Usage]
- “Diagnostic tracing scopes” [SQL Anywhere Server - SQL Usage]
- “sa_diagnostic_tracing_level table” on page 1081
- “Diagnostic tracing” [SQL Anywhere Server - SQL Usage]

**Examples**

The following example sets the tracing level to 1. The entire database is profiled for performance counter data, and some samples of executed statements:

```sql
CALL sa_set_tracing_level( 1 );
```

The following example sets the tracing level to 3, and specifies the user AG84756. Only activities associated with AG84756 are traced:

```sql
CALL sa_set_tracing_level( 3, 'user', 'AG84756' );
```
sa_snapshots system procedure

Returns a list of snapshots that are currently active.

Syntax

sa_snapshots( )

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection_num</td>
<td>INTEGER</td>
<td>The connection ID for the connection on which the snapshot is running.</td>
</tr>
<tr>
<td>start_sequence_num</td>
<td>UNSIGNED BIGINT</td>
<td>A unique number that identifies the snapshot.</td>
</tr>
<tr>
<td>statement_level</td>
<td>BIT</td>
<td>True if the snapshot was created with statement-snapshot or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>readonly-statement-snapshot. Otherwise, false.</td>
</tr>
</tbody>
</table>

Remarks

Several statement snapshots can exist on one connection. For nested or interleaved statements running under statement snapshot isolation levels, each one begins a different statement snapshot with its first read or update.

Usually there is only one transaction snapshot per connection (one entry per connection in sa_snapshots with statement_level=0). However, a snapshot associated with a cursor never changes after the cursor's first fetch and a cursor opened WITH HOLD stays open through a commit or rollback. If the cursor has an associated snapshot, then the snapshot also persists. Therefore, it is possible for multiple transaction snapshots to exist for the same connection_num: one for the current transaction snapshot and one or more for old transaction snapshots that persist because of WITH HOLD cursors.

Privileges

You must have the MONITOR system privilege.

Side effects

None

See also

- “sa_transactions system procedure” on page 1261
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]

Example

You can execute the following query to identify snapshots that are currently active.

```sql
CALL sa_snapshots( );
```
**sa_split_list system procedure**

Takes a string of values, separated by a delimiter, and returns a set of rows—one row for each value.

**Syntax**

```sql
sa_split_list(
    str
    [,
    delim
    [,
    maxlen ] ]
)
```

**Arguments**

- **str** Use this LONG VARCHAR parameter to specify the string containing the values to be split, separated by `delim`.
- **delim** Use this optional CHAR(10) parameter to specify the delimiter used in `str` to separate values. The delimiter can be a string of any characters, up to 10 bytes. If `delim` is not specified, a comma is used by default.
- **maxlen** Use this optional INTEGER parameter to specify the maximum length of the returned values. For example, if `maxlen` is set to 3, the values in the result set are truncated to a length of 3 characters. If you specify 0 (the default), values can be any length.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>line_num</td>
<td>INTEGER</td>
<td>Sequential number for the row.</td>
</tr>
<tr>
<td>row_value</td>
<td>LONG VARCHAR</td>
<td>Value from the string, truncated to <code>maxlen</code> if required.</td>
</tr>
</tbody>
</table>

**Remarks**

The `sa_split_list` procedure accepts a string with a delimited list of values, and returns a result set with one value per row. This is the opposite of the action performed by the LIST function [Aggregate]. An empty string is returned for `row_value` if the string:

- begins with `delim`
- contains two successive instances of `delim` in the middle of the string
- ends with `delim`

White space within the input string is significant. If the delimiter is a space character, extra spaces in the input string result in extra rows in the result set. If the delimiter is not a space character, spaces in the input string are not trimmed from the values in the result set.

**Privileges**

None

**Side effects**

None
See also

- “LIST function [Aggregate]” on page 285

Examples

The following query returns a list of black colored products.

```sql
SELECT list(Name) 
FROM Products
WHERE Color = 'Black';
```

<table>
<thead>
<tr>
<th>list (Products.Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tee Shirt, Baseball Cap, Visor, Shorts</td>
</tr>
</tbody>
</table>

In the following example, the sa_split_list procedure is used to return the original result set from the aggregated list.

```sql
SELECT * 
FROM sa_split_list( 'Tee Shirt, Baseball Cap, Visor, Shorts' );
```

<table>
<thead>
<tr>
<th>line_num</th>
<th>row_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tee Shirt</td>
</tr>
<tr>
<td>2</td>
<td>Baseball Cap</td>
</tr>
<tr>
<td>3</td>
<td>Visor</td>
</tr>
<tr>
<td>4</td>
<td>Shorts</td>
</tr>
</tbody>
</table>

The following example returns a row for each word. To avoid returning rows where row_value is an empty string, the WHERE clause must be specified.

```sql
SELECT * 
FROM sa_split_list( 'one||three|four||six|', '|' )
WHERE row_value <> '';
```

<table>
<thead>
<tr>
<th>line_num</th>
<th>row_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>one</td>
</tr>
<tr>
<td>3</td>
<td>three</td>
</tr>
<tr>
<td>4</td>
<td>four</td>
</tr>
<tr>
<td>6</td>
<td>six</td>
</tr>
</tbody>
</table>

In the following example, a procedure called ProductsWithColor is created. When called, the ProductsWithColor procedure uses sa_split_list to parse the color values specified by the user, looks in the Color column of the Products table, and returns the name, description, size, and color for each product that matches one of the user-specified colors.
The result of the procedure call below is the name, description, size, and color of all products that are either white or black.

```
CREATE OR REPLACE PROCEDURE ProductsWithColor( IN color_list LONG VARCHAR )
BEGIN
    SELECT Name, Description, Size, Color
    FROM Products
    WHERE Color IN ( SELECT row_value FROM sa_split_list( color_list ) );
END;
```

```
SELECT * from ProductsWithColor( 'white,black' );
```

**sa_stack_trace system procedure**

Returns the stack trace leading to the current call location.

**Syntax**

```
sa_stack_trace()
```

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StackLevel</td>
<td>UNSIGNED SMALLINT</td>
<td>Current line has Stack Level 1.</td>
</tr>
<tr>
<td>UserName</td>
<td>CHAR(128)</td>
<td>The name of the owner of the procedure or trigger, or NULL if the current level is in a batch.</td>
</tr>
<tr>
<td>ProcName</td>
<td>CHAR(128)</td>
<td>The name of the procedure or trigger where the call was performed or the batch type.</td>
</tr>
<tr>
<td>LineNumber</td>
<td>UNSIGNED INTEGER</td>
<td>The line number of the call within the procedure, trigger, or batch.</td>
</tr>
</tbody>
</table>

**Remarks**

Each record in the result set represents a single call on the stack. If the compound statement is not part of a procedure, function, trigger, or event, the type of batch (watcom_batch or tsql_batch) is returned instead of the procedure name.

This procedure returns the same information as the STACK_TRACE function.

**Privileges**

None

**Side effects**

None.
See also

- “TRY statement” on page 1022
- “BEGIN statement” on page 523
- “ERROR_LINE function [Miscellaneous]” on page 228
- “ERROR_MESSAGE function [Miscellaneous]” on page 229
- “ERROR_PROCEDURE function [function type]” on page 230
- “ERROR_SQLCODE function [Miscellaneous]” on page 231
- “ERROR_SQLSTATE function [Miscellaneous]” on page 232
- “ERROR_STACK_TRACE function [Miscellaneous]” on page 233
- “STACK_TRACE function [Miscellaneous]” on page 371
- “sa_error_stack_trace system procedure” on page 1136
- “Nested compound statements and exception handlers” [SQL Anywhere Server - SQL Usage]

Example

This example shows how to obtain the result set columns from the sa_stack_trace system procedure:

```sql
SELECT StackLevel, UserName, ProcName, LineNumber FROM sa_stack_trace();
```

When this statement is executed outside of the context of a stored procedure, the result set is empty.

The following example shows the implementation of a general stack trace procedure that sends its results to the client window. An example of its use is included.

```sql
CREATE OR REPLACE PROCEDURE StackDump( MSG CHAR(128) )
BEGIN
    DECLARE myStackLevel UNSIGNED SMALLINT;
    DECLARE myUserName CHAR(128);
    DECLARE myProcName CHAR(128);
    DECLARE myLineNumber UNSIGNED SMALLINT;
    DECLARE err_notfound EXCEPTION FOR SQLSTATE '02000';
    DECLARE myStack CURSOR FOR
        SELECT StackLevel, UserName, ProcName, LineNumber FROM sa_stack_trace();

    MESSAGE 'Stack Trace: ' || MSG TO CLIENT;
    OPEN myStack;
    StackLoop: LOOP
        FETCH NEXT myStack INTO myStackLevel, myUserName, myProcName, myLineNumber;
        IF SQLSTATE = err_notfound THEN
            LEAVE StackLoop;
        END IF;
        IF myStackLevel != 1 THEN
            MESSAGE myStackLevel - 1 || ' ' || myUserName || ' ' || myProcName || ' ' || myLineNumber TO CLIENT;
        ENDIF
    END LOOP StackLoop;
    CLOSE myStack;
END;
```

```sql
CREATE OR REPLACE PROCEDURE Proc1()
BEGIN
    CALL Proc2();
END;
```

```sql
CREATE OR REPLACE PROCEDURE Proc2()
```
BEGIN
  CALL Proc3();
END;

CREATE OR REPLACE PROCEDURE Proc3()
BEGIN
  CALL StackDump('Snapshot from Proc3');
END;

CALL Proc1();

The Interactive SQL Messages window output is shown below.

Stack Trace: Snapshot from proc3
1 DBA proc3 3
2 DBA proc2 3
3 DBA proc1 3

---

**sa_statement_text system procedure**

Formats a SELECT statement so that individual items appear on separate lines. This is useful when viewing long statements from the request log, in which all newline characters are removed.

**Syntax**

```
sa_statement_text( txt )
```

**Arguments**

- **txt** Use this LONG VARCHAR parameter to specify a SELECT statement.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stmt_text</td>
<td>LONG VARCHAR</td>
<td>A clause of the SELECT statement.</td>
</tr>
</tbody>
</table>

**Remarks**

The text that is entered must be a string (in single quotes) or a string expression.

**Privileges**

None

**Side effects**

None

**See also**

- “sa_get_request_times system procedure” on page 1149
- “sa_get_request_profile system procedure” on page 1148
Example

The following call formats a SELECT statement so that individual items appear on separate lines.

```sql
CALL sa_statement_text( 'SELECT * FROM car WHERE name=''Audi''' );
```

<table>
<thead>
<tr>
<th>stmt_text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SELECT *</td>
</tr>
<tr>
<td>2 FROM car</td>
</tr>
</tbody>
</table>
| 3 WHERE name = 'Audi'

**sa_table_fragmentation system procedure**

Reports information about the fragmentation of database tables.

**Syntax**

```sql
sa_table_fragmentation(
    [ tbl_name
    [, owner_name ] ]
)
```

**Arguments**

- **tbl_name** Use this optional CHAR(128) parameter to specify the name of the table to check for fragmentation. The default is NULL.

- **owner_name** Use this optional CHAR(128) parameter to specify the owner of `tbl_name`. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableName</td>
<td>CHAR(128)</td>
<td>Name of the table.</td>
</tr>
<tr>
<td>rows</td>
<td>UNSIGNED INTEGER</td>
<td>Number of rows in the table.</td>
</tr>
<tr>
<td>row_segments</td>
<td>UNSIGNED BIGINT</td>
<td>Number of row segments in the table.</td>
</tr>
<tr>
<td>segs_per_row</td>
<td>DOUBLE</td>
<td>Number of segments per row.</td>
</tr>
</tbody>
</table>

**Remarks**

Database administrators can use this procedure to obtain information about the fragmentation in a database's tables. If no arguments are supplied, results are returned for all tables in the database.

When database tables become excessively fragmented, you can run REORGANIZE TABLE or rebuild the database to reduce table fragmentation and improve performance.
Privileges

You must have the MANAGE ANY STATISTICS or MONITOR system privilege.

Side effects

None

See also

- “Reduce table fragmentation” [SQL Anywhere Server - SQL Usage]
- “Database rebuilds” [SQL Anywhere Server - SQL Usage]
- “REORGANIZE TABLE statement” on page 935

Examples

```sql
CALL sa_table_fragmentation( 'Products','GROUPO' );
CALL sa_table_fragmentation( owner_name='GROUPO' );
```

sa_table_page_usage system procedure

Reports information about the page usage of database tables.

Syntax

```sql
sa_table_page_usage( )
```

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableId</td>
<td>UNSIGNED INTEGER</td>
<td>The table ID.</td>
</tr>
<tr>
<td>TablePages</td>
<td>INTEGER</td>
<td>The number of table pages used by the table.</td>
</tr>
<tr>
<td>PctUsedT</td>
<td>INTEGER</td>
<td>The percentage of used table page space.</td>
</tr>
<tr>
<td>IndexPages</td>
<td>INTEGER</td>
<td>The number of index pages used by the table.</td>
</tr>
<tr>
<td>PctUsedI</td>
<td>INTEGER</td>
<td>The percentage of used index page space.</td>
</tr>
<tr>
<td>PctOfFile</td>
<td>INTEGER</td>
<td>The percentage of the total database file the table occupies.</td>
</tr>
<tr>
<td>TableName</td>
<td>CHAR(128)</td>
<td>The table name.</td>
</tr>
</tbody>
</table>

Remarks

The results include the same information provided by the Information utility. When the progress_messages database option is set to Raw or Formatted, progress messages are sent from the database server to the client while the sa_table_page_usage system procedure is running.
Privileges
You must have the MANAGE ANY DBSPACE system privilege.

Side effects
None

See also
- “Information utility (dbinfo)” [SQL Anywhere Server - Database Administration]
- “progress_messages option” [SQL Anywhere Server - Database Administration]

Example
The following example obtains information about the page usage of the SalesOrderItems table.

```
SELECT * FROM sa_table_page_usage()
WHERE TableName = 'SalesOrderItems';
```

**sa_table_stats system procedure**
Reports information about how many pages have been read from each table.

**Syntax**
```
sa_table_stats()
```

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>INTEGER</td>
<td>The table ID.</td>
</tr>
<tr>
<td>creator</td>
<td>CHAR(128)</td>
<td>The user name of the table's creator.</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>The table name.</td>
</tr>
<tr>
<td>&quot;count&quot;</td>
<td>UNSIGNED BIGINT</td>
<td>The estimated number of rows in the table, taken from SYSTAB.</td>
</tr>
<tr>
<td>table_page_count</td>
<td>UNSIGNED BIGINT</td>
<td>The number of main pages used by the table.</td>
</tr>
<tr>
<td>table_page_cached</td>
<td>UNSIGNED BIGINT</td>
<td>The number of tables pages currently stored in the cache.</td>
</tr>
<tr>
<td>table_page_reads</td>
<td>UNSIGNED BIGINT</td>
<td>The number of page reads performed for pages in the main table.</td>
</tr>
<tr>
<td>ext_page_count</td>
<td>UNSIGNED BIGINT</td>
<td>The estimated number of pages in the table</td>
</tr>
<tr>
<td>ext_page_cached</td>
<td>UNSIGNED BIGINT</td>
<td>Column reserved for future use.</td>
</tr>
<tr>
<td>ext_page_reads</td>
<td>UNSIGNED BIGINT</td>
<td>Column reserved for future use.</td>
</tr>
</tbody>
</table>
Remarks

Each row returned by the sa_table_stats procedure describes a table for which the optimizer is maintaining page statistics. The sa_table_stats procedure can be used to find which tables are using cache memory and how many disk reads are being performed for each table. For example, you can use the sa_table_stats procedure to find the table that is generating the most disk reads. The results of the procedure represent estimates and should be used only for diagnostic purposes.

The table_page_cached column indicates how many pages of the table are currently stored in the cache, and the table_page_reads column indicates how many table pages have been read from disk since the optimizer started maintaining counts for the table. These statistics are not stored persistently within the database; they represent the activity on tables after they are loaded into memory for the first time.

Privileges

You must have the MONITOR system privilege.

Side effects

None

See also

- “SYSTAB system view” on page 1399

Example

You can execute the following query to get information about how many pages have been read from each table.

```
CALL sa_table_stats();
```

**sa_text_index_stats system procedure**

Returns statistical information about the text indexes in the database.

Syntax

```
sa_text_index_stats()
```

Remarks

Use the sa_text_index_stats system procedure to view statistical information for each text index in the database. The following table describes the information returned by sa_text_index_stats.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner_id</td>
<td>UNSIGNED INTEGER</td>
<td>ID of the owner of the table</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INTEGER</td>
<td>ID of the table</td>
</tr>
<tr>
<td>index_id</td>
<td>UNSIGNED INTEGER</td>
<td>ID of the text index</td>
</tr>
<tr>
<td>Column name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>text_config_id</td>
<td>UNSIGNED BIGINT</td>
<td>ID of the text configuration object referenced by the index</td>
</tr>
<tr>
<td>owner_name</td>
<td>CHAR(128)</td>
<td>Name of the owner</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>Name of the table</td>
</tr>
<tr>
<td>index_name</td>
<td>CHAR(128)</td>
<td>Name of the text index</td>
</tr>
<tr>
<td>text_config_name</td>
<td>CHAR(128)</td>
<td>Name of the text configuration object</td>
</tr>
<tr>
<td>doc_count</td>
<td>UNSIGNED BIGINT</td>
<td>Total number of indexed column values in the text index</td>
</tr>
<tr>
<td>doc_length</td>
<td>UNSIGNED BIGINT</td>
<td>Total length of data in the text index</td>
</tr>
<tr>
<td>pending_length</td>
<td>UNSIGNED BIGINT</td>
<td>Total length of the pending changes</td>
</tr>
<tr>
<td>deleted_length</td>
<td>UNSIGNED BIGINT</td>
<td>Total length of the pending deletions</td>
</tr>
<tr>
<td>last_refresh</td>
<td>TIMESTAMP</td>
<td>Local date and time of the last refresh</td>
</tr>
</tbody>
</table>

The pending_length, deleted_length, and last_refresh values are 0 for IMMEDIATE REFRESH text indexes.

For MANUAL REFRESH text indexes, you can use doc_length, pending_length, and deleted_length to decide whether to refresh the text index, and the type of refresh to perform (rebuild vs. incremental).

**Privileges**

You must have one of the following system privileges:

- MANAGE ANY STATISTICS
- CREATE ANY INDEX
- ALTER ANY INDEX
- DROP ANY INDEX
- CREATE ANY OBJECT
- ALTER ANY OBJECT
- DROP ANY OBJECT

**Side effects**

None
See also

- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Text index concepts and reference” [SQL Anywhere Server - SQL Usage]
- “DROP TEXT INDEX statement” on page 782
- “REFRESH TEXT INDEX statement” on page 928
- “TRUNCATE TEXT INDEX statement” on page 1021
- “sa_refresh_text_indexes system procedure” on page 1211
- “sa_text_index_vocab system procedure” on page 1258
- “SYSTEXTIDX system view” on page 1407

Example

The following statement returns statistical information for each text index in the database:

```
CALL sa_text_index_stats( );
```

**sa_text_index_vocab system procedure**

Lists all terms that appear in a CHAR text index, and the total number of indexed values that each term appears in.

**Syntax**

```
sa_text_index_vocab(indexname, tabname [, tabowner ])
```

**Arguments**

- **indexname** Use this CHAR(128) parameter to specify the name of the text index.
- **tabname** Use this CHAR(128) parameter to specify the name of the table on which the text index is built.
- **tabowner** Use this optional CHAR(128) parameter to specify the owner of the table. The default is NULL.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term</td>
<td>VARCHAR(60)</td>
<td>A term in the text index.</td>
</tr>
<tr>
<td>freq</td>
<td>BIGINT</td>
<td>The number of indexed values the term appears in.</td>
</tr>
</tbody>
</table>

**Remarks**

The sa_text_index_vocab system procedure returns all terms that appear in a text index, and the total number of indexed values that each term appears in (which is less than the total number of occurrences if the term appears multiple times in some indexed values).
Privileges

You must have SELECT privilege on the indexed table, or have the SELECT ANY TABLE system privilege.

Side effects

None

See also

- “sa_text_index_vocab_nchar system procedure” on page 1259
- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Term and phrase search” [SQL Anywhere Server - SQL Usage]
- “DROP TEXT INDEX statement” on page 782
- “REFRESH TEXT INDEX statement” on page 928
- “TRUNCATE TEXT INDEX statement” on page 1021
- “sa_refresh_text_indexes system procedure” on page 1211
- “SYSTEXTIDX system view” on page 1407

Example

The following example builds a text index called VocabTxtIdx on the Products.Description column in the sample database.

    CREATE TEXT INDEX VocabTxtIdx2 ON Products( Description );

The next statement executes the sa_text_index_vocab system procedure to return all the terms that appear in the text index.

    SELECT * FROM sa_text_index_vocab( 'VocabTxtIdx2', 'Products', 'GROUPO' );

<table>
<thead>
<tr>
<th>term</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cap</td>
<td>2</td>
</tr>
<tr>
<td>Cloth</td>
<td>1</td>
</tr>
<tr>
<td>Cotton</td>
<td>2</td>
</tr>
<tr>
<td>Crew</td>
<td>1</td>
</tr>
<tr>
<td>Hooded</td>
<td>1</td>
</tr>
<tr>
<td>neck</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

sa_text_index_vocab_nchar system procedure

Lists all terms that appear in an NCHAR text index, and the total number of indexed values that each term appears in.
Syntax

```sql
sa_text_index_vocab_nchar(indexname, tabname [, tabowner ])
```

Arguments

- **indexname**  Use this CHAR(128) parameter to specify the name of the text index.
- **tabname**  Use this CHAR(128) parameter to specify the name of the table on which the text index is built.
- **tabowner**  Use this optional CHAR(128) parameter to specify the owner of the table. The default in NULL.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>term</td>
<td>NVARCHAR(60)</td>
<td>A term in the text index.</td>
</tr>
<tr>
<td>freq</td>
<td>BIGINT</td>
<td>The number of indexed values the term appears in.</td>
</tr>
</tbody>
</table>

Remarks

The `sa_text_index_vocab_nchar` system procedure returns all terms that appear in a text index, and the total number of indexed values that each term appears in (which is less than the total number of occurrences if the term appears multiple times in some indexed values).

Privileges

You must have SELECT privilege on the indexed table, or have the SELECT ANY TABLE system privilege.

Side effects

None

See also

- “sa_text_index_vocab system procedure” on page 1258
- “Full text search” [SQL Anywhere Server - SQL Usage]
- “Term and phrase search” [SQL Anywhere Server - SQL Usage]
- “DROP TEXT INDEX statement” on page 782
- “REFRESH TEXT INDEX statement” on page 928
- “TRUNCATE TEXT INDEX statement” on page 1021
- “sa_refresh_text_indexes system procedure” on page 1211
- “SYSTEXTIDX system view” on page 1407
**Example**

The following example builds a text index called VocabNTxtIdx on the NProducts.Description column. The NProducts table is a version of the Products table with NCHAR columns instead of CHAR columns.

```sql
CREATE TEXT INDEX VocabNTxtIdx2 ON NProducts( Description );
```

The next statement executes the `sa_text_index_vocab_nchar` system procedure to return all the terms that appear in the text index.

```sql
SELECT * FROM sa_text_index_vocab_nchar( 'VocabNTxtIdx2', 'NProducts' );
```

<table>
<thead>
<tr>
<th>term</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cap</td>
<td>2</td>
</tr>
<tr>
<td>Cloth</td>
<td>1</td>
</tr>
<tr>
<td>Cotton</td>
<td>2</td>
</tr>
<tr>
<td>Crew</td>
<td>1</td>
</tr>
<tr>
<td>Hooded</td>
<td>1</td>
</tr>
<tr>
<td>neck</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**sa_transactions system procedure**

Returns a list of transactions that are currently active.

**Syntax**

```sql
sa_transactions()
```

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection_num</td>
<td>INTEGER</td>
<td>The connection ID for the connection the transaction is running on.</td>
</tr>
<tr>
<td>transaction_id</td>
<td>INTEGER</td>
<td>The ID that uniquely identifies the transaction as long as the database server keeps track of it. IDs are reused as old transaction information is discarded.</td>
</tr>
<tr>
<td>start_time</td>
<td>TIMESTAMP</td>
<td>The TIMESTAMP for when the transaction started.</td>
</tr>
</tbody>
</table>
### Remarks

This procedure provides information about the transactions that are currently running against the database.

### Privileges

You must have the MONITOR system privilege.

### Side effects

None

### See also

- “sa_snapshots system procedure” on page 1247
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]

### Example

You can execute the following query to identify transactions that are currently active.

```sql
CALL sa_transactions();
```

### sa_unload_cost_model system procedure

Unloads the current cost model to the specified file.

#### Syntax

```sql
sa_unload_cost_model( file_name )
```

#### Arguments

- **file_name** Use this CHAR(256) parameter to specify the name of the file in which to unload the data. Because it is the database server that executes the system procedure, `file_name` specifies a file on the database server computer, and a relative `file_name` specifies a file relative to the database server's starting directory.
Remarks
The optimizer uses cost models to determine optimal access plans for queries. The database server maintains a cost model for each database. The cost model for a database can be recalibrated at any time using the CALIBRATE SERVER clause of the ALTER DATABASE statement. For example, you might decide to recalculate the cost model if you move the database onto non-standard hardware.

The sa_unload_cost_model system procedure allows you to save a cost model to an ASCII file (file_name). You can then log into another database and use the sa_load_cost_model system procedure to load the cost model from the first database into the second one. This avoids having to recalculate the second database.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>The sa_unload_cost_model system procedure does not include CALIBRATE PARALLEL READ information in the file.</td>
</tr>
</tbody>
</table>

Using the sa_unload_cost_model system procedure eliminates repetitive, time-consuming recalibration activities when there is a large number of similar hardware installations.

You must have write permissions where the file is created.

Privileges
You must have the SELECT ANY TABLE system privilege.

Side effects
None

See also
- “ALTER DATABASE statement” on page 426
- “sa_load_cost_model system procedure” on page 1167
- “Advanced: Query optimization” [SQL Anywhere Server - SQL Usage]

Example
The following example unloads the cost model to a file called costmodel8:

```
CALL sa_unload_cost_model( 'costmodel8' );
```

**sa_user_defined_counter_add system procedure**
Adapts the value of a user-defined counter by a specified amount.

**Syntax**
```
sa_user_defined_counter_add(
    counter_name,
    [ delta ],
    [ apply_to_con ],
    [ apply_to_db ],
    [ apply_to_server ] )
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.01263
Arguments

- **counter_name** Use this VARCHAR(128) parameter to specify the name of the user-defined counter whose value you want to change. Examples of user-defined counter names are UserDefinedCounterRate01 and UserDefinedCounterRaw01.

- **delta** Use this BIGINT parameter to specify the amount that the user-defined counter is incremented or decremented by. The default is 1.

- **apply_to_con** Use this INTEGER parameter to specify whether to adjust the counter value for the current connection. 0 means do not adjust the value, and 1 means adjust the value. The default is 1.

- **apply_to_db** Use this INTEGER parameter to specify whether to adjust the counter value for the database. 0 means do not adjust the value, and 1 means adjust the value. The default is 1.

- **apply_to_server** Use this INT parameter to specify whether to adjust the counter value for the database server. 0 means do not adjust the value, and 1 means adjust the value. The default is 1.

Returns

This function returns an INTEGER status code.

Remarks

This function returns 1 if *delta* is defined, 0 if *delta* is not defined, and an error code if an error occurs. Examples of errors include:

- an invalid counter name
- an invalid value for the apply_to_server, apply_to_db, or apply_to_con parameter.

Concurrent access to counters is applied atomically, so a counter value can be incremented from multiple, concurrent requests.

User-defined counters are implemented as 32-bit UNSIGNED INTEGER values.

Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

None

See also

- “sa_user_defined_counter_set system procedure” on page 1265
- “List of connection properties” [SQL Anywhere Server - Database Administration]
- “List of database properties” [SQL Anywhere Server - Database Administration]
- “List of database server properties” [SQL Anywhere Server - Database Administration]
- “User-defined statistics” [SQL Anywhere Server - SQL Usage]
- “User-defined properties” [SQL Anywhere Server - Database Administration]
Example

The following statement increments the value of UserDefinedCounterRate01 by 2 for the current
connection, database, and database server:

```
SELECT sa_user_defined_counter_add( 'UserDefinedCounterRate01', 2, 1, 1, 1 );
```

**sa_user_defined_counter_set** system procedure

Sets a user-defined counter to a specified value.

**Syntax**

```
sa_user_defined_counter_set(
  counter_name
, value
[, apply_to_con
[, apply_to_db
[, apply_to_server ] ] ]
)
```

**Arguments**

- **counter_name**  Use this VARCHAR(128) parameter to specify the name of the user-defined
counter whose value you want to change. Examples of user-defined counter names are
UserDefinedCounterRate01 and UserDefinedCounterRaw01.

- **value**  Use this BIGINT parameter to specify the value to which the user-defined counter is set.

- **apply_to_con**  Use this INT parameter to specify whether to adjust the counter value for the
current connection. 0 means do not adjust the value, and 1 means adjust the value. The default is 1.

- **apply_to_db**  Use this INT parameter to specify whether to adjust the counter value for the
database. 0 means do not adjust the value, and 1 means adjust the value. The default is 0.

- **apply_to_server**  Use this INT parameter to specify whether to adjust the counter value for the
database server. 0 means do not adjust the value, and 1 means adjust the value. The default is 0.

**Returns**

This function returns an INTEGER status code.

**Remarks**

This function returns 1 if value is defined, 0 if value is not defined, and an error code if an error occurs.
Examples of errors include:

- an invalid counter name
- an invalid value for the apply_to_server, apply_to_db, or apply_to_con parameter.

Concurrent access to counters is applied atomically, so a counter value can be reset from multiple,
concurrent requests.

User-defined counters are implemented as 32-bit UNSIGNED INTEGER values.
Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

None

See also

- “sa_user_defined_counter_add system procedure” on page 1263
- “List of connection properties” [SQL Anywhere Server - Database Administration]
- “List of database properties” [SQL Anywhere Server - Database Administration]
- “List of database server properties” [SQL Anywhere Server - Database Administration]
- “User-defined properties” [SQL Anywhere Server - Database Administration]
- “User-defined statistics” [SQL Anywhere Server - SQL Usage]

Example

The following statement sets the value of UserDefinedCounterRate01 to 0 for the current connection, database, and database server:

```
SELECT sa_user_defined_counter_set( 'UserDefinedCounterRate01', 0, 1, 1, 1 );
```

**sa_validate system procedure**

Performs a checksum validation on all, or parts, of a database.

Syntax

```
sa_validate(
    [ tbl_name ]
    [ , owner_name ]
)
```

Arguments

- **tbl_name** Use this optional CHAR(128) parameter to specify the name of a table or materialized view to validate. The default is NULL.

- **owner_name** Use this optional CHAR(128) parameter to specify an owner. When specified by itself, all tables and materialized views owned by the owner are validated. The default is NULL.

Privileges

You must have the VALIDATE ANY OBJECT system privilege.

Side effects

None
Remarks

<table>
<thead>
<tr>
<th>Argument specified</th>
<th>Type of validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>All tables, materialized views, and indexes in the database are validated. The database itself is also validated, including checksum validation.</td>
</tr>
<tr>
<td>tbl_name</td>
<td>The specified table, or materialized view, and all of its indexes, that are owned by the current user are validated.</td>
</tr>
<tr>
<td>owner_name</td>
<td>All tables, materialized views, and indexes owned by the specified user are validated.</td>
</tr>
<tr>
<td>tbl_name and owner_name</td>
<td>The specified table, or materialized view, and all of its indexes, that are owned by the specified user are validated.</td>
</tr>
</tbody>
</table>

The procedure returns a single column named Messages. Errors returned during validation appear in the column. If validation succeeds without error, the column contains No error detected.

Caution
Validating a table or an entire database should be performed while no connections are making changes to the database; otherwise, errors may be reported indicating some form of database corruption even though no corruption actually exists.

Example
The following statement performs a validation of tables and materialized views owned by user DBA:

```sql
CALL sa_validate( owner_name = 'DBA' );
```

sa_verify_password system procedure

Validates the password of the current user.

Syntax

```sql
sa_verify_password( curr_pswd )
```

Arguments

- `curr_pswd`  Use this CHAR(128) parameter to specify the password of the current database user.

Returns

The function returns an INTEGER value.

Remarks

This procedure is used by sp_password. If the password matches, 0 is returned and no error occurs. If the password does not match, an error is diagnosed. The connection is not terminated if the password does not match.
Privileges
None

Side effects
None

See also
● “Adaptive Server Enterprise system procedures” on page 1090

Example
The following example attempts to validate the current connection's password when the current user is DBA or User1. An error occurs if the current password does not match.

```sql
IF USER_NAME() = 'DBA' THEN
    SELECT sa_verify_password('sql');
ELSEIF USER_NAME() = 'User1' THEN
    SELECT sa_verify_password('user');
END IF;
```

sp_alter_secure_feature_key system procedure
Alters a previously defined secure feature key by modifying the authorization key and/or the feature list.

Syntax
```sql
sp_alter_secure_feature_key(
    name
  , auth_key
  , features
)
```

Arguments
● **name**  The VARCHAR (128) name for the secure feature key you want to alter. A key with the given name must already exist.

● **auth_key**  The CHAR (128) case-sensitive authorization key for the secure feature key. The authorization key must be either a non-empty string of at least six characters, or NULL, indicating that the existing authorization key is not to be changed.

● **features**  The LONG VARCHAR, comma-separated list of secure features that the key can enable. If feature_list is NULL, the existing feature_list is not changed.

Remarks
This procedure allows you to alter the authorization key or feature list of an existing secure feature key.

Privileges
You must have the SERVER OPERATOR system privilege and have the MANAGE KEYS feature enabled on the connection.
Side effects

None

See also

- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_create_secure_feature_key system procedure” on page 1273
- “sp_drop_secure_feature_key system procedure” on page 1279
- “sp_list_secure_feature_keys system procedure” on page 1288
- “sp_use_secure_feature_key system procedure” on page 1321
- List of secure features: “-sf database server option” [SQL Anywhere Server - Database Administration]

Example

In order for the following example to work, the server must be started with the option: -sk securefkey.

This example enables the SYSTEM secure feature set which includes MANAGE KEYS, creates a new secure feature key called MYSET with case-sensitive authorization key securemyset, alters which secure features belong to the set, and then uses sp_list_secure_feature_keys to obtain a list of the currently defined secure feature keys:

```sql
CALL sp_use_secure_feature_key( 'system', 'securefkey' );
CALL sp_create_secure_feature_key( 'myset', 'securemyset', 'local' );
CALL sp_alter_secure_feature_key( 'myset', 'securemyset', 'local,remote' );
CALL sp_list_secure_feature_keys( );
```

sp_auth_sys_role_info system procedure

Returns the mapping of authorities from previous versions of SQL Anywhere to their corresponding compatibility roles.

Syntax

```
sp_auth_sys_role_info( )
```

Result set

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth</td>
<td>VARCHAR(20)</td>
<td>The name of the authority.</td>
</tr>
<tr>
<td>role_name</td>
<td>CHAR(128)</td>
<td>The name of the corresponding compatibility role.</td>
</tr>
<tr>
<td>role_id</td>
<td>UNSIGNED INTEGER</td>
<td>The ID number for the compatibility role.</td>
</tr>
</tbody>
</table>

Remarks

None
Privileges

None

Standards and compatibility

- SQL/2008  Vendor extension.

See also

- “Compatibility roles” [SQL Anywhere Server - Database Administration]
- “Authorities become compatibility roles” [SQL Anywhere Server - Database Administration]

Example

The following statement returns the list of authorities (auth) from pre-16.0 SQL Anywhere versions, mapped to their corresponding compatibility roles (role_name) provided as of version 16.0.

```sql
CALL sp_auth_sys_role_info();
```

<table>
<thead>
<tr>
<th>auth</th>
<th>role_name</th>
<th>role_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA</td>
<td>SYS_AUTH_DBA_ROLE</td>
<td>2147485648</td>
</tr>
<tr>
<td>RESOURCE</td>
<td>SYS_AUTH_RESOURCE_ROLE</td>
<td>2147485649</td>
</tr>
<tr>
<td>BACKUP</td>
<td>SYS_AUTH_BACKUP_ROLE</td>
<td>2147485650</td>
</tr>
<tr>
<td>VALIDATE</td>
<td>SYS_AUTH_VALIDATE_ROLE</td>
<td>2147485651</td>
</tr>
<tr>
<td>READFILE</td>
<td>SYS_AUTH_READFILE_ROLE</td>
<td>2147485652</td>
</tr>
<tr>
<td>PROFILE</td>
<td>SYS_AUTH_PROFILE_ROLE</td>
<td>2147485653</td>
</tr>
<tr>
<td>READCLIENTFILE</td>
<td>SYS_AUTH_READCLIENTFILE_ROLE</td>
<td>2147485654</td>
</tr>
<tr>
<td>WRITECLIENTFILE</td>
<td>SYS_AUTH_WRITECLIENTFILE_ROLE</td>
<td>2147485655</td>
</tr>
<tr>
<td>WRITEFILE</td>
<td>SYS_AUTH_WRITEFILE_ROLE</td>
<td>2147485656</td>
</tr>
<tr>
<td>REMOTE DBA</td>
<td>SYS_RUN_REPLICATION_ROLE</td>
<td>2147485664</td>
</tr>
</tbody>
</table>

**sp_copy_directory** system procedure

Copies a directory to a specified location.

**Syntax**

```sql
sp_copy_directory(
    source_path
    , destination_path
)
```
Parameters

- **source_path** Use this LONG NVARCHAR parameter to specify the path of the directory to copy. The path can be absolute or relative. A relative path is resolved on the computer relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

- **destination_path** Use this LONG NVARCHAR parameter to specify the path where the directory is copied to. The path can be absolute or relative. A relative path is resolved on the computer relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located. The directory is created if it does not already exist.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function copies the directory and its files from the source directory to the specified directory. The directory and its files remain in the source directory. Use the sp_delete_directory system procedure to delete the source directory.

Privileges

You must have the READ FILE and WRITE FILE system privileges.

See also

- “sp_copy_file system procedure” on page 1271
- “sp_create_directory system procedure” on page 1272
- “sp_delete_directory system procedure” on page 1275
- “sp_delete_file system procedure” on page 1275
- “sp_list_directory system procedure” on page 1286
- “sp_move_directory system procedure” on page 1289
- “sp_move_file system procedure” on page 1291
- “Directory and file system procedures” on page 1088

Example

The following statement makes a copy of the subdirectories and files in the *SQLAnywhere* directory in *SQLAnywhere.bkp*.

```sql
SELECT sp_copy_directory('c:\sqlany\samples\SQLAnywhere', 'c:\sqlany\samples\SQLAnywhere.bkp');
```

The entire directory including its subdirectories and files are duplicated.

**sp_copy_file system procedure**

Copies a file to a specified location.
Syntax

    sp_copy_file(
        source_path,
        destination_path
    )

Parameters

- **source_path** Use this LONG NVARCHAR parameter to specify the file path, including file name, of the file to move. The path can be absolute or relative. A relative path is resolved on the computer relative to the current working directory of the database server. When the sandbox feature is enabled, then absolute and relative paths refer to the directory where the main database file is located.

- **destination_path** Use this LONG NVARCHAR parameter to specify the new directory of the file. The path can be absolute or relative. A relative path is resolved on the computer relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function copies a file from one directory to another. The file remains in the source directory. You can delete the file in the source directory using the sp_delete_file system procedure.

Privileges

You must have the READ FILE and WRITE FILE system privileges.

See also

- “sp_copy_directory system procedure” on page 1270
- “sp_create_directory system procedure” on page 1272
- “sp_delete_directory system procedure” on page 1275
- “sp_delete_file system procedure” on page 1275
- “sp_list_directory system procedure” on page 1286
- “sp_move_directory system procedure” on page 1289
- “sp_move_file system procedure” on page 1291
- “Directory and file system procedures” on page 1088

Example

The following statement copies the file license.txt to the c:\temp directory and gives it a new name.

    SELECT sp_copy_file('c:\sqlany\license.txt', 'c:\temp\license.bkp');

A duplicate copy of the license.txt file exists in the temp directory under a different name.

**sp_create_directory system procedure**

Creates a directory on the computer.
Syntax

```
sp_create_directory( root_path )
```

Parameters

- `root_path` Use this LONG NVARCHAR parameter to specify the directory path to create. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function creates a new directory.

Privileges

You must have the WRITE FILE system privilege.

See also

- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_copy_directory system procedure” on page 1270
- “sp_copy_file system procedure” on page 1271
- “sp_delete_directory system procedure” on page 1275
- “sp_delete_file system procedure” on page 1275
- “sp_list_directory system procedure” on page 1286
- “sp_move_directory system procedure” on page 1289
- “sp_move_file system procedure” on page 1291
- “Directory and file system procedures” on page 1088

Example

The following statement creates a directory named `SQLAnywhere.bkp` in the `c:\sqlany\samples` directory.

```
SELECT sp_create_directory( 'c:\sqlany\samples\SQLAnywhere.bkp' );
```

**sp_create_secure_feature_key system procedure**

Creates a new secure feature key.

Syntax

```
sp_create_secure_feature_key(
    name
    , auth_key
    , features
)
```
Arguments

- **name**  The VARCHAR (128) name for the new secure feature key. This argument cannot be NULL or an empty string.

- **auth_key**  The CHAR (128) case-sensitive authorization key for the secure feature key. The authorization key must be a non-empty string of at least six characters.

- **features**  The LONG VARCHAR comma-separated list of secure features that the new key can enable.

  Specifying - before a feature means that the feature is not re-enabled when the secure feature key is set.

Remarks

This procedure creates a new secure feature key that can be given to any user. The SYSTEM secure feature key is created by using the -sk database server option.

Privileges

You must have the SERVER OPERATOR system privilege and have the MANAGE_KEYS feature enabled on the connection.

Side effects

None

See also

- List of secure features: “-sf database server option” [SQL Anywhere Server - Database Administration]
- “-sk database server option” [SQL Anywhere Server - Database Administration]
- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_alter_secure_feature_key system procedure” on page 1268
- “sp_drop_secure_feature_key system procedure” on page 1279
- “sp_list_secure_feature_keys system procedure” on page 1288
- “sp_use_secure_feature_key system procedure” on page 1321

Example

In order for the following example to work, the server must be started with the option: -sk securefkey.

This example enables the SYSTEM secure feature set which includes MANAGE_KEYS, creates a new secure feature key called MYSET with case-sensitive authorization key securemyset, and then uses sp_list_secure_feature_keys to obtain a list of the currently defined secure feature keys:

```sql
CALL sp_use_secure_feature_key( 'system', 'securefkey' );
CALL sp_create_secure_feature_key( 'myset', 'securemyset', 'local' );
CALL sp_list_secure_feature_keys( );
```
sp_delete_directory system procedure

Deletes the specified directory.

Syntax

```sql
sp_delete_directory( root_path )
```

Parameters

- `root_path` Use this LONG NVARCHAR parameter to specify the directory path of the directory to be deleted. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function deletes a directory from a specified location.

Privileges

You must have the READ FILE and WRITE FILE system privileges.

See also

- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_copy_directory system procedure” on page 1270
- “sp_copy_file system procedure” on page 1271
- “sp_create_directory system procedure” on page 1272
- “sp_delete_file system procedure” on page 1275
- “sp_list_directory system procedure” on page 1286
- “sp_move_directory system procedure” on page 1289
- “sp_move_file system procedure” on page 1291
- “Directory and file system procedures” on page 1088

Example

The following statement deletes the directory `SQLAnywhere.bkp`.

```sql
SELECT sp_delete_directory('c:\sqlany\samples\SQLAnywhere.bkp');
```

sp_delete_file system procedure

Deletes the specified file from the computer.

Syntax

```sql
sp_delete_file( file_path )
```
Parameters

- **file_path** Use this LONG NVARCHAR parameter to specify the file path, including file name, of the file to be deleted. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function deletes the specified file from the computer.

You must be connected to a database server on the computer where the file resides to run this procedure.

Privileges

You must have the READ FILE and WRITE FILE system privileges.

See also

- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_copy_directory system procedure” on page 1270
- “sp_copy_file system procedure” on page 1271
- “sp_create_directory system procedure” on page 1272
- “sp_delete_directory system procedure” on page 1275
- “sp_list_directory system procedure” on page 1286
- “sp_move_directory system procedure” on page 1289
- “sp_move_file system procedure” on page 1291
- “Directory and file system procedures” on page 1088

Example

The following statement deletes the license.bkp file from the temp directory.

```
SELECT sp_delete_file('c:\temp\license.bkp');
```

**sp_displayroles system procedure**

Returns all roles granted to the specified system privilege, system role, user-defined role, or user name, or displays the entire hierarchy tree of roles.

Syntax

```
sp_displayroles(
    user_role_name,
    display_mode,
    grant_type
)
```
Arguments

- **user_role_name** Use this CHAR(128) parameter to specify the name of a system privilege, system role, user-defined role, or user name. If it is not specified or is NULL, then the current user is used by default.

- **display_mode** Use this VARCHAR(30) parameter to specify whether to return parent-level or child-level hierarchy, relative to **user_role_name**. If **display_mode** is not specified or is NULL, then only explicitly granted roles and privileges are returned (no inherited roles or privileges). Possible values for **display_mode** include the following:
  - **expand_up** Shows the system roles granted to **user_role_name** in the parent hierarchy tree for **user_role_name**.
  - **expand_down** Shows the system roles and privileges granted to **user_role_name**, including the role hierarchy tree for the child levels of **user_role_name**.

- **grant_type** Use this VARCHAR(30) parameter to control the grant type returned. If it is not specified, then ALL is used by default. Possible values for **grant_type** including the following:
  - **NO_ADMIN** Shows all roles and system privileges granted to **user_role_name** with the WITH NO ADMIN OPTION or WITH ADMIN OPTION clause.
  - **ADMIN** Returns all roles and system privileges granted to **user_role_name** with the WITH ADMIN OPTION or WITH ADMIN ONLY OPTION clause.
  - **ALL** Shows all roles/system privileges granted to **user_role_name**.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_name</td>
<td>CHAR(128)</td>
<td>The role or system privilege granted to <strong>user_role_name</strong>.</td>
</tr>
<tr>
<td>parent_role_name</td>
<td>CHAR(128)</td>
<td>The role names for the parents of <strong>user_role_name</strong>.</td>
</tr>
<tr>
<td>grant_type</td>
<td>CHAR(10)</td>
<td>Information about whether <strong>user_role_name</strong> has administrative rights. Possible values: NO ADMIN, ADMIN, or ADMIN ONLY.</td>
</tr>
<tr>
<td>role_level</td>
<td>SMALLINT</td>
<td>With expand_down mode: Level is 1 for directly granted roles, 2 for the next level below, and so on.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>With expand_up mode: Level is 0 for the roles to which <strong>user_role_name</strong> has been granted, -1 for the next hierarchy above, and so on.</td>
</tr>
</tbody>
</table>

Remarks

For system privileges, the result shows the system privilege name instead of the system privilege role name. With expand_down mode, the parent_role_name is NULL for level 1 (directly granted roles). With the default mode, the role_level column is 1 and parent_role_name is NULL, since with default mode only the directly granted roles are displayed.
If this procedure is used for a user with mode expand_up, then no results are returned since a user resides at the top level in any role hierarchy. Similarly, if this procedure is used for an immutable system privilege, with mode expand_down, then no results are returned because an immutable system privilege resides at the bottom level in any role hierarchy. The default mode is to display only the directly granted roles/system privileges.

**Standards and compatibility**

- **SQL/2008**  
  Vendor extension.

**Privileges**

No privileges are required to execute this procedure on yourself. However, to return the system privileges or roles for another user ID or a role, you must have the MANAGE ROLES system privilege.

**Side effects**

None

**See also**

- “Roles” [SQL Anywhere Server - Database Administration]
- “Viewing the roles and privileges for a user or role (SQL)” [SQL Anywhere Server - Database Administration]
- “Viewing the roles and privileges for a user or role (Sybase Central)” [SQL Anywhere Server - Database Administration]
- “sp_sys_priv_role_info system procedure” on page 1311
- “sp_has_role system procedure” on page 1284
- “sp_proc_priv system procedure” on page 1297
- “sp_objectpermission system procedure” on page 1292

**Example**

The following statement returns all roles granted to the user issuing the command.

```sql
CALL sp_displayroles();
```

This examples returns the list of system privileges granted to the SYS_SPATIAL_ADMIN_ROLE system role:

```sql
CALL sp_displayroles( 'SYS_SPATIAL_ADMIN_ROLE' );
```

<table>
<thead>
<tr>
<th>role_name</th>
<th>parent_role_name</th>
<th>grant_type</th>
<th>role_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUBLIC</td>
<td>(NULL)</td>
<td>NO ADMIN</td>
<td>1</td>
</tr>
<tr>
<td>MANAGE ANY SPATIAL OBJECT</td>
<td>(NULL)</td>
<td>NO ADMIN</td>
<td>1</td>
</tr>
</tbody>
</table>

This examples returns the list of system privileges granted to the SYS_SPATIAL_ADMIN_ROLE, including all roles above it in the hierarchy of roles:

```sql
CALL sp_displayroles( 'SYS_SPATIAL_ADMIN_ROLE', 'expand_up' );
```
<table>
<thead>
<tr>
<th>role_name</th>
<th>parent_role_name</th>
<th>grant_type</th>
<th>role_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_AUTH_DBA_ROLE</td>
<td>dbo</td>
<td>ADMIN</td>
<td>-3</td>
</tr>
<tr>
<td>SYS_AUTH_DBA_ROLE</td>
<td>SYS_RUN_REPLICATION_ROLE</td>
<td>ADMIN</td>
<td>-3</td>
</tr>
<tr>
<td>SYS_AUTH_SSO_ROLE</td>
<td>SYS_AUTH_DBA_ROLE</td>
<td>ADMIN</td>
<td>-2</td>
</tr>
<tr>
<td>MANAGE ROLES</td>
<td>SYS_AUTH_SSO_ROLE</td>
<td>ADMIN</td>
<td>-1</td>
</tr>
<tr>
<td>MANAGE ROLES</td>
<td>SYS_REPLICATION_ADMIN_ROLE</td>
<td>NO ADMIN</td>
<td>-1</td>
</tr>
<tr>
<td>SYS_SPATIAL_ADMIN_ROLE</td>
<td>MANAGE ROLES</td>
<td>ADMIN ONLY</td>
<td>0</td>
</tr>
</tbody>
</table>

The following statement returns all system privileges granted to the user User1:

```sql
CALL sp_displayroles( 'User1' );
```

This example returns the list of system privileges pertaining to views:

```sql
SELECT sys_priv_name FROM sp_sys_priv_role_info()
WHERE sys_priv_name LIKE '%VIEW%'
```

---

### sp_drop_secure_feature_key system procedure

Deletes a secure feature key.

#### Syntax

```sql
sp_drop_secure_feature_key( name )
```

#### Arguments

- **name** The VARCHAR (128) name of the secure feature key to drop.

#### Remarks

If the named key does not exist, an error is returned. If the named key exists, it is deleted as long as it is not the last secure feature key that is allowed to manage secure features and secure feature keys. The SYSTEM secure feature key cannot be dropped unless there is another key that has the MANAGE_FEATURES and MANAGE_KEYS secure features enabled.

#### Privileges

You must have the SERVER OPERATOR system privilege and have the MANAGE_KEYS feature enabled on the connection.

#### Side effects

None
See also
- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_alter_secure_feature_key system procedure” on page 1268
- “sp_create_secure_feature_key system procedure” on page 1273
- “sp_list_secure_feature_keys system procedure” on page 1288
- “sp_use_secure_feature_key system procedure” on page 1321
- MANAGE_FEATURES and MANAGE_KEYS: “-sf database server option” [SQL Anywhere Server - Database Administration]

Example
In order for the following example to work, the server must be started with the option: -sk securefkey.

This example enables the SYSTEM secure feature set which includes MANAGE_KEYS and then drops the secure feature key called MYSET:

```
CALL sp_use_secure_feature_key( 'system', 'securefkey' );
CALL sp_drop_secure_feature_key( 'myset' );
```

**sp_forward_to_remote_server system procedure**

Syntax
```
sp_forward_to_remote_server(  
   @server_name , @sql  
)
```

Arguments
- **@server_name** Use this CHAR(128) parameter to specify the name of the remote server the SQL statement is executed on.
- **@sql** Use this LONG VARCHAR parameter to specify the SQL statement to execute on the remote server.

Remarks
This procedure allows an application to execute a SQL statement on a remote server and retrieve any result sets generated by that statement. The SQL statement is sent verbatim to the remote server and therefore SQL Anywhere does not need to able to parse the statement.

To use this system procedure, you must define the remote server with the CREATE SERVER statement.

Unlike the FORWARD TO statement, sp_forward_to_remote_server can be used within procedures. However, this stored procedure cannot be used within the FROM clause of a SELECT statement since the schema of the remote result sets is arbitrary. You can fetch remote result sets by declaring a cursor on a stored procedure that is called in the sp_forward_to_remote_server procedure.
Note
If the SQL statement returns multiple result sets, the sp_forward_to_remote_server stored procedure returns each remote result set in turn.

Privileges
None

Side effects
There are no local side effects to executing this stored procedure; however, since the SQL statement that is executed on the remote server is arbitrary, there may be side effects on the remote server.

See also
● “FORWARD TO statement” on page 808
● “CREATE SERVER statement” on page 657

Example
The following example uses the sp_forward_to_remote_server stored procedure to determine the number of non-system tables in a remote SQL Anywhere 16 database called RemoteSA.

    CALL sp_forward_to_remote_server( 'RemoteSA',
        'SELECT COUNT(*) FROM sys.systable WHERE CREATOR NOT IN (0,3,6)' );

The following example uses the sp_forward_to_remote_server stored procedure to read the columns from an Excel spreadsheet named newSalesData.

    call sp_forward_to_remote_server( 'RemoteExcel', 'SELECT * FROM newSalesData' );

sp_get_last_synchronize_result system procedure
Returns information about the last synchronization initiated by the SYNCHRONIZE statement.

Syntax

    sp_get_last_synchronize_result(
        [ @conn_id ]
        [, @complete_only ] ]
    )

Arguments

● @conn_id Use this optional INTEGER parameter to specify the connection ID number for a connection on which the SYNCHRONIZE statement was executed. The default is NULL. If the parameter is not specified or is NULL, then the connection ID of the current connection is used.

● @complete_only Set this optional BIT parameter to 1 to have the stored procedure return information about completed synchronizations. Set the parameter to 0 to return information about synchronizations that are currently active. The default is 1.
Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_id</td>
<td>BIGINT</td>
<td>The primary key of the table used to determine the order in which rows were inserted into the table.</td>
</tr>
<tr>
<td>conn_id</td>
<td>UNSIGNED INTEGER</td>
<td>The connection id number of the connection that executed the SYNCHRONIZE statement that generated this event.</td>
</tr>
<tr>
<td>result_time</td>
<td>TIMESTAMP</td>
<td>The time the event was added to the synchronize_results table.</td>
</tr>
<tr>
<td>result_type</td>
<td>CHAR(128)</td>
<td>The type of event.</td>
</tr>
<tr>
<td>parm_id</td>
<td>INTEGER</td>
<td>Each event can have zero or more parameters associated with it. The parm_id column orders the parameters associated with each event.</td>
</tr>
<tr>
<td>parm_result</td>
<td>LONG VARCHAR</td>
<td>The message text associated with the event parameter.</td>
</tr>
</tbody>
</table>

Remarks

To view details of past or current synchronizations, you can use the sp_get_last_synchronize_result stored procedure as an alternative to directly querying the synchronize_results and synchronize_parameters global shared temporary table. The stored procedure only returns the results of the last synchronization for the specified connection ID number. If you do not specify any parameters, the last completed synchronization on the current connection is returned.

You can also use this stored procedure to monitor the progress of a synchronization on a connection that is different from your current connection. To monitor the progress of a synchronization on a different connection:

1. Execute a SELECT CONNECTIONPROPERTY statement to determine the connection ID of your current connection.

2. Execute a SYNCHRONIZE statement using the connection ID returned by the SELECT CONNECTIONPROPERTY statement.

3. On a different connection, execute a SELECT CONNECTIONPROPERTY statement and set the complete_only parameter to 0. Information about the last synchronization for the specified connection is returned, even if the synchronization is incomplete.

Following is a list of events and their associated parm_id values from the synchronize_parameters table:

<table>
<thead>
<tr>
<th>Event</th>
<th>parm_id value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBSC_EVENTTYPE_ERROR_MSG</td>
<td>0</td>
<td>The text of the error message.</td>
</tr>
<tr>
<td>Event</td>
<td>parm_id value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>DBSC_EVENTTYPE_WARNING_MSG</td>
<td>0</td>
<td>The text of the warning message.</td>
</tr>
<tr>
<td>DBSC_EVENTTYPE_INFO_MSG</td>
<td>0</td>
<td>The text of the information message.</td>
</tr>
<tr>
<td>DBSC_EVENTTYPE_PROGRESS_INDEX</td>
<td>0</td>
<td>The new progress index value.</td>
</tr>
<tr>
<td>DBSC_EVENTTYPE_PROGRESS_TEXT</td>
<td>0</td>
<td>The new progress text.</td>
</tr>
<tr>
<td>DBSC_EVENTTYPE_TITLE</td>
<td>0</td>
<td>The new window title.</td>
</tr>
<tr>
<td>DBSC_EVENTTYPE_SYNC_DONE</td>
<td>0</td>
<td>The exit code from the synchronization. 0 indicates success.</td>
</tr>
<tr>
<td>DBSC_EVENTTYPE_ML_CONNECT</td>
<td>0</td>
<td>The communications protocol being used.</td>
</tr>
<tr>
<td>DBSC_EVENTTYPE_DOWNLOAD_COMMITTED</td>
<td>0</td>
<td>The number of insert/update operations committed.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>The number of delete operations committed.</td>
</tr>
<tr>
<td>DBSC_EVENTTYPE_UPLOAD_SENT</td>
<td>0</td>
<td>The number of insert operations uploaded.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>The number of update operations uploaded.</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>The number of delete operations uploaded.</td>
</tr>
</tbody>
</table>

**Privileges**

You must have SELECT privilege on the synchronize_results and synchronize_parameters shared global temporary tables.
Side effects
None

See also
- “SYNCHRONIZE statement [MobiLink]” on page 1011
- “DBSC_Event structure [Dbmlsync .NET]” [MobiLink - Client Administration]

Example
The following example returns information about the last synchronization that completed on the current connection.

```sql
CALL sp_get_last_synchronize_result();
```

The following example uses named parameters when calling the sp_get_last_synchronize_result system procedure, and returns information about the last completed synchronization that was initiated from connection ID 25.

```sql
CALL sp_get_last_synchronize_result(
    @conn_id=25,
    @complete_only=1);
```

sp_has_role system procedure
Returns whether the invoker of the procedure has been granted the specified system privilege or user-defined role.

Syntax
```sql
sp_has_role(
    rolename
    [
        , grant_type
        [, throw_error ]
    ]
)
```

Parameters
- **rolename** Use this CHAR(128) parameter to specify the name of a system privilege or user-defined role to check for.

- **grant_type** Use this optional CHAR(20) parameter to specify the grant type to check for. Possible values are: ADMIN, or NO ADMIN (the default).

  If set to ADMIN, the sp_has_role procedure checks whether the invoking user has administrative rights for the privilege or role. If set to NO ADMIN, sp_has_role checks whether the invoking user has the rights to exercise the privilege or role.

- **throw_error** Use this optional INTEGER parameter to specify whether to return a value indicating the outcome of the privilege check. If 1 is specified, a message is returned if the invoker has not been granted the specified privilege or role. If 0 is specified (the default), no message is returned, regardless.
Returns

<table>
<thead>
<tr>
<th>Value returned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The system privilege or user-defined role is granted to invoking user, and the grant_type check passes as well.</td>
</tr>
<tr>
<td>0 or Permission denied: you do not have permission to execute this command/procedure.</td>
<td>The system privilege or user-defined role is not granted to invoking user, and/or the grant_type check failed as well. The error message replaces the value 0 when throw_error is set to 1.</td>
</tr>
<tr>
<td>-1</td>
<td>The system privilege or user-defined role specified does not exist. No error message appears, even if throw_error is set to 1.</td>
</tr>
</tbody>
</table>

Remarks

The throw_error argument is useful for returning "permission denied" error messages when a user fails the permission check in a stored procedure.

Privileges

None

See also

- “Roles” [SQL Anywhere Server - Database Administration]
- “sp_sys_priv_role_info system procedure” on page 1311
- “sp_proc_priv system procedure” on page 1297
- “sp_displayroles system procedure” on page 1276
- “sp_objectpermission system procedure” on page 1292

Standards and compatibility

- SQL/2008  Vendor extension.

Example

- The following statement checks whether you have been granted the CREATE ANY PROCEDURE system privilege with no administrative rights.

  ```sql
  SELECT sp_has_role( 'CREATE ANY PROCEDURE' );
  ```

- The following statement checks whether you have been granted the CREATE ANY PROCEDURE system privilege with administrative rights and returns an error only if you have not been granted it.

  ```sql
  SELECT sp_has_role( 'CREATE ANY PROCEDURE', 'ADMIN', 1 );
  ```

- The following statement checks whether you have been granted Role_A with no administrative rights.

  ```sql
  SELECT sp_has_role( 'Role_A' );
  ```

- The following statement checks whether you have been granted Role_A with administrative rights and returns an error only if you have not been granted it.
sp_list_directory system procedure

Returns information about the contents of a directory.

Syntax

```sql
sp_list_directory(
  root_path
  [, max_depth ]
)
```

Parameters

- **root_path**  Use this LONG NVARCHAR parameter to specify the absolute or relative directory path. The relative path is relative to the working directory of the server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

- **max_depth**  Use this optional INTEGER parameter to specify the maximum number of directories to traverse. A max_depth of NULL, 0, or a negative value results in all subdirectories of root_path being traversed. The default is NULL.

Result set

This procedure returns information about the files and subdirectories in a specified directory. This procedure returns a table with the following columns:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_path</td>
<td>LONG NVARCHAR - The path to a file or subdirectory within the specified directory. If directory_path is specified as a relative path, then the returned file_path value is relative. Otherwise, the file_path value is absolute. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.</td>
</tr>
<tr>
<td>file_type</td>
<td>NVARCHAR(1) - F if the file_path value is a file or D when the file_path value is a directory.</td>
</tr>
<tr>
<td>file_size</td>
<td>UNSIGNED BIGINT - Specifies the size in bytes of the file or NULL when file_path value is a directory</td>
</tr>
<tr>
<td>owner</td>
<td>NVARCHAR(128) - The owner of the file or directory.</td>
</tr>
<tr>
<td>create_date_time</td>
<td>TIMESTAMP WITH TIME ZONE - The date and time the file or directory was created, returned in the database server's timezone.</td>
</tr>
</tbody>
</table>
### Column name | Column description
---|---
modified_date_time | TIMESTAMP WITH TIME ZONE - The date and time the file or directory was last modified, returned in the database server's timezone.
access_date_time | TIMESTAMP WITH TIME ZONE - The date and time the file or directory was last accessed, returned in the database server's timezone.
permissions | VARCHAR(10) - The set of access permissions for the file or directory.

**Remarks**

The max_depth parameter specifies the maximum depth that the directory will be traversed. A max_depth of NULL, 0, or a negative value implies that there is no restriction.

**Privileges**

You must have the READ FILE system privilege.

**Standards and compatibility**

- [SQL/2008](#) Vendor extension.

**See also**

- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_copy_directory system procedure” on page 1270
- “sp_copy_file system procedure” on page 1271
- “sp_create_directory system procedure” on page 1272
- “sp_delete_directory system procedure” on page 1275
- “sp_delete_file system procedure” on page 1275
- “sp_move_directory system procedure” on page 1289
- “sp_move_file system procedure” on page 1291
- “Directory and file system procedures” on page 1088

**Examples**

The following statement lists the files and subdirectories of the `c:\sqlany\samples\SQLAnywhere` directory.

```sql
CALL sp_list_directory('c:\sqlany\samples\SQLAnywhere');
```

The following statement returns the total number of files and folders in the `c:\sqlany\samples\SQLAnywhere` and `c:\sqlany\samples\UltraLite` directories.

```sql
SELECT COUNT(*) FROM sp_list_directory( 'c:\sqlany\samples\SQLAnywhere' )
UNION ALL
SELECT COUNT(*) FROM sp_list_directory( 'c:\sqlany\samples\UltraLite' );
```

If both the `c:\sqlany\samples\SQLAnywhere` and `c:\sqlany\samples\UltraLite` directories contain the same file names, file types, and file sizes, then the following statement combines the results from two directories and gives you the total number of unique results.
SELECT COUNT(*) FROM
( SELECT REPLACE( file_path, 'c:\sqlany\samples\SQLAnywhere', '' ) AS file_path, file_type, file_size
  FROM sp_list_directory( 'c:\sqlany\samples\SQLAnywhere' )
  UNION
  SELECT REPLACE( file_path, 'c:\sqlany\samples\UltraLite', '' ) AS file_path, file_type, file_size
  FROM sp_list_directory( 'c:\sqlany\samples\UltraLite' ) ) d;

sp_list_secure_feature_keys system procedure

Returns a list of defined secure feature keys.

Syntax

sp_list_secure_feature_keys( )

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>The name of the secure feature key.</td>
</tr>
<tr>
<td>features</td>
<td>LONG VARCHAR</td>
<td>The secure features enabled by the secure feature key.</td>
</tr>
</tbody>
</table>

Remarks

This procedure returns the names of existing secure feature keys, as well as the set of secure features that can be enabled by each key.

If the user has the MANAGE_FEATURES and MANAGE_KEYS secure features enabled, then the procedure returns a list of all secure feature keys.

If the user only has the MANAGE_KEYS secure feature enabled, then the procedure returns keys that have the same features or a subset of the same features that the current user has enabled.

Privileges

You must have the SERVER OPERATOR system privilege, and have the MANAGE_KEYS feature enabled on the connection.

Side effects

None

See also

- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_alter_secure_feature_key system procedure” on page 1268
- “sp_create_secure_feature_key system procedure” on page 1273
- “sp_drop_secure_feature_key system procedure” on page 1279
- “sp_use_secure_feature_key system procedure” on page 1321
Example

In order for the following example to work, the server must be started with the option: `-sk securefkey`.

This example enables the SYSTEM secure feature set which includes the MANAGE_FEATURES and MANAGE_KEYS secure features, and then uses `sp_list_secure_feature_keys` to obtain a list of all the currently defined secure feature keys:

```sql
CALL sp_use_secure_feature_key( 'system', 'securefkey' );
CALL sp_list_secure_feature_keys( );
```

### sp_login_environment system procedure

Sets connection options when users log in.

**Syntax**

```sql
sp_login_environment();
```

**Remarks**

`sp_login_environment` is the default procedure called by the `login_procedure` database option.

It is recommended that you do not edit this procedure. Instead, to change the login environment, set the `login_procedure` option to point to a different procedure.

**Privileges**

None

**Side effects**

None

**See also**

- “`login_procedure option`” [SQL Anywhere Server - Database Administration]

### sp_move_directory system procedure

This function moves the directory pointed to by `source_path` to the destination pointed to by `destination_path`.

**Syntax**

```sql
sp_move_directory(
    source_path
,   destination_path
)
```
Parameters

- **source_path**  Use this LONG NVARCHAR parameter to specify the path of the directory to be moved. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

- **destination_path**  Use this LONG NVARCHAR parameter to specify the path where the directory is to be moved to. The directory is created if it does not already exist. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

Returns

This function returns 0 on success and 1 on failure.

Remarks

This function moves all the files in the source directory to the specified directory, and then deletes the source directory.

Privileges

You must have the READ FILE and WRITE FILE system privileges.

See also

- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_copy_directory system procedure” on page 1270
- “sp_copy_file system procedure” on page 1271
- “sp_create_directory system procedure” on page 1272
- “sp_delete_directory system procedure” on page 1275
- “sp_delete_file system procedure” on page 1275
- “sp_list_directory system procedure” on page 1286
- “sp_move_file system procedure” on page 1291
- “Directory and file system procedures” on page 1088

Examples

The following statement moves the SQLAnywhere directory including its files and subdirectories to \temp\SQLAnywhere.

```sql
SELECT sp_move_directory('c:\sqlany\samples\SQLAnywhere', 'c:\temp\SQLAnywhere');
```

The original directory is removed.

The following statement restores the SQLAnywhere directory to its original location.

```sql
SELECT sp_move_directory('c:\temp\SQLAnywhere', 'c:\sqlany\samples\SQLAnywhere');
```
**sp_move_file system procedure**

Moves the specified file to a new directory on the computer.

**Syntax**

```sql
sp_move_file(
    source_path,
    destination_path
)
```

**Parameters**

- **source_path**  Use this LONG NVARCHAR parameter to specify the file path, including file name, of the file to be moved. The path can be absolute or relative. A relative path is resolved to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

- **destination_path**  Use this LONG NVARCHAR parameter to specify the destination file path, including file name, on the computer. The path can be absolute or relative. A relative path is resolved relative to the current working directory of the database server. When the sandbox feature is enabled, the absolute and relative paths refer to the directory where the main database file is located.

**Returns**

This function returns 0 on success and 1 on failure.

**Remarks**

This function moves a file from one directory to another.

**Privileges**

You must have the READ FILE and WRITE FILE system privileges.

**See also**

- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_copy_directory system procedure” on page 1270
- “sp_copy_file system procedure” on page 1271
- “sp_create_directory system procedure” on page 1272
- “sp_delete_directory system procedure” on page 1275
- “sp_delete_file system procedure” on page 1275
- “sp_list_directory system procedure” on page 1286
- “sp_move_directory system procedure” on page 1289
- “Directory and file system procedures” on page 1088

**Examples**

The following statement moves the file `license.txt` to the `c:\temp` directory and gives it a new name.

```sql
SELECT sp_move_file('c:\sqlany\license.txt', 'c:\temp\license.bkp');
```

The file no longer exists in its original location.
The following statement restores the file to its original name and location.

```sql
SELECT sp_move_file('c:\temp\license.bkp', 'c:\sqlany\license.txt');
```

### sp_objectpermission system procedure

Generates a report on object privileges granted to a specified role or user ID, or a report on object privileges granted on a specified object or dbspace.

#### Syntax

```sql
sp_objectpermission(
  object_name
  , object_owner
  , object_type
)
```

#### Arguments

- **object_name**  The CHAR(128) name of an object or dbspace or a user or a role. If this argument is not specified, the object privileges of the current user are displayed. Default value is NULL.

- **object_owner** The CHAR(128) name of the object owner for the specified object name. The object privileges of the specified object owned by the specified object owner are displayed. Default value is NULL.

- **object_type** The CHAR(20) type of database object. If no value is specified, privileges on all object types are returned. The default value is NULL. Valid values are:
  - DBSPACE
  - FUNCTION
  - MATERIALIZED VIEW
  - PROCEDURE
  - SEQUENCE
  - TABLE (Column-level object privileges are also displayed)
  - USER
  - VIEW

#### Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>CHAR(128)</td>
<td>Returns the user ID of the grantor.</td>
</tr>
<tr>
<td>grantee</td>
<td>CHAR(128)</td>
<td>Returns the user ID of the grantee.</td>
</tr>
<tr>
<td>object_name</td>
<td>CHAR(128)</td>
<td>Returns the name of the object.</td>
</tr>
<tr>
<td>owner</td>
<td>CHAR(128)</td>
<td>Returns the name of the object owner.</td>
</tr>
<tr>
<td>object_type</td>
<td>CHAR(20)</td>
<td>Returns the type of object.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>column_name</td>
<td>CHAR(128)</td>
<td>Returns the name of the column.</td>
</tr>
<tr>
<td>permission</td>
<td>CHAR(20)</td>
<td>Returns the name of the privilege.</td>
</tr>
<tr>
<td>grantable</td>
<td>CHAR(1)</td>
<td>Returns a value indicating whether the privilege is grantable.</td>
</tr>
</tbody>
</table>

**Remarks**

All arguments are optional and can generate the following reports:

- If the input is an object (table, view, procedure, function, sequence, and so on), then the procedure displays a list of all users and roles that have different object privilege on the object.

- If the input is a role or user, then the procedure displays a list of all object privileges granted to the role or input.

- If the input is a dbspace name, then the procedure displays a list of all users and roles which has CREATE privilege on the specified dbspace.

When executing sp_objectpermission to display object privileges for a user or a role, the object privileges that are inherited through role grants are also displayed. By default the object_type is NULL and the object privileges for all existing object types matching the specified object name are displayed.

**Standards and compatibility**

- **SQL/2008** Vendor extension.

**Privileges**

- No privileges are required to execute this procedure on yourself or on objects you own. However, to call this procedure on another user ID, or on an object owned by another user ID, you must have the MANAGE ANY OBJECT PRIVILEGE system privilege.

- To execute this procedure for a dbspace, you must have the MANAGE ANY DBSPACE system privilege.

**Side effects**

None
See also

- “Roles” [SQL Anywhere Server - Database Administration]
- “Viewing the roles and privileges for a user or role (SQL)” [SQL Anywhere Server - Database Administration]
- “Viewing the roles and privileges for a user or role (Sybase Central)” [SQL Anywhere Server - Database Administration]
- “sp_sys_priv_role_info system procedure” on page 1311
- “sp_has_role system procedure” on page 1284
- “sp_proc_priv system procedure” on page 1297
- “sp_displayroles system procedure” on page 1276

Example

The following statement returns the object-level privileges granted to the DIAGNOSTICS system role. For the purposes of this example, the results have been truncated.

```
CALL sp_objectpermission( 'diagnostics' );
```

<table>
<thead>
<tr>
<th>grantor</th>
<th>grantees</th>
<th>object_name</th>
<th>owner</th>
<th>object_type</th>
<th>column_name</th>
<th>permission</th>
<th>grantable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>diagnostics</td>
<td>sa_tmp_diagnostic_optre-write</td>
<td>dbo</td>
<td>TABLE</td>
<td>(NULL)</td>
<td>SELECT</td>
<td>N</td>
</tr>
<tr>
<td>SYS</td>
<td>diagnostics</td>
<td>sa_tmp_diagnostic_optre-write</td>
<td>dbo</td>
<td>TABLE</td>
<td>(NULL)</td>
<td>INSERT</td>
<td>N</td>
</tr>
<tr>
<td>SYS</td>
<td>diagnostics</td>
<td>sa_tmp_diagnostic_optre-write</td>
<td>dbo</td>
<td>TABLE</td>
<td>(NULL)</td>
<td>UPDATE</td>
<td>N</td>
</tr>
<tr>
<td>SYS</td>
<td>diagnostics</td>
<td>sa_tmp_diagnostic_optre-order</td>
<td>dbo</td>
<td>TABLE</td>
<td>(NULL)</td>
<td>DELETE</td>
<td>N</td>
</tr>
<tr>
<td>SYS</td>
<td>diagnostics</td>
<td>sa_tmp_diagnostic_optre-order</td>
<td>dbo</td>
<td>TABLE</td>
<td>(NULL)</td>
<td>SELECT</td>
<td>N</td>
</tr>
<tr>
<td>SYS</td>
<td>diagnostics</td>
<td>sa_tmp_diagnostic_optre-order</td>
<td>dbo</td>
<td>TABLE</td>
<td>(NULL)</td>
<td>INSERT</td>
<td>N</td>
</tr>
<tr>
<td>SYS</td>
<td>diagnostics</td>
<td>sa_tmp_diagnostic_optre-order</td>
<td>dbo</td>
<td>TABLE</td>
<td>(NULL)</td>
<td>UPDATE</td>
<td>N</td>
</tr>
</tbody>
</table>
The results show that there are many tables on which the DIAGNOSTICS system role has object-level permissions. This result makes sense because the tables are used for storing diagnostic information for database tracing and application profiling.

The following statement returns the object-level privileges granted to the ml_server user.

```
CALL sp_objectpermission( 'ml_server' );
```

The following statement returns the object-level privileges on the system dbspace.

```
CALL sp_objectpermission( object_name='system', object_type='DBSPACE' );
```

**sp_parse_json system procedure**

Returns a representation of JSON data using SQL data types.

**Syntax**

```
sp_parse_json(
    var
    , "json"
    [, maxlen ]
)
```

**Arguments**

- **var** Use this LONG VARCHAR parameter to specify the name of the local variable to create. The type of variable is determined at execution time. The variable is created if it does not exist.

- **"json"** Use this LONG NVARCHAR parameter to specify a string representation of the JSON data structure.

- **maxlen** Use this INTEGER parameter to specify the limit of recursion. Nodes below the stipulated depth are not processed. Instead, they are returned as a fragment of JSON.
Remarks

This procedure processes a JSON object and returns the processed data as a SQL data type. The type of return variable is determined when the procedure is executed. In most cases, the return variables are either ROW or ARRAY SQL data types.

The sp_parse_json system procedure returns a VARCHAR fragment for its base ARRAY or OBJECT when subsequent instances return a different number of nodes.

JSON object identifiers must comply with the identifier rules defined in the database server. As well, the database server enforces the same limits for JSON data types as it does for the underlying ROW and ARRAY data types.

Privileges

None

Side effects

None

See also

- “Composite data types” on page 128
- “Identifiers” on page 4
- “ROW constructor [Composite]” on page 351
- “ARRAY constructor [Composite]” on page 157

Example

The following example sets up a table with some data to generate a JSON string (\["name":"Frank","age":51},{"name":"Bill","age":22},
{"name":"Jackie","age":37}\]) that can be parsed. It then parses the JSON string into a SQL array data type.

BEGIN
   DECLARE json_data LONG VARCHAR;
   CREATE LOCAL TEMPORARY TABLE test (name AS VARCHAR(64),
               age AS INT);

   INSERT INTO test (name, age) VALUES ('Frank',51);
   INSERT INTO test (name, age) VALUES ('Bill',22);
   INSERT INTO test (name, age) VALUES ('Jackie',37);

   SELECT * INTO json_data FROM test FOR JSON RAW;

   CALL sp_parse_json ( 'sql_array', json_data );

   SELECT sql_array [[row_num]].name AS name, sql_array [[row_num]].age AS age
           FROM sa_rowgenerator ( 1, 3 );
END;
sp_proc_priv system procedure

Returns the list of system privileges required to run a procedure.

Syntax

```
sp_proc_priv(  
    ['proc_name']  
)
```

Arguments

- **proc_name**  This CHAR(128) parameter specifies the name of the procedure to return privileges for. If *proc_name* is not specified, the privileges required for all procedures are returned. The default is NULL.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc_name</td>
<td>CHAR(128)</td>
<td>The name of the procedure.</td>
</tr>
<tr>
<td>privilege</td>
<td>LONG VARCHAR</td>
<td>The list of privileges required to run the procedure.</td>
</tr>
</tbody>
</table>

Remarks

In the result set, if a number of privileges separated by a comma is listed for a stored procedure, then it implies that any one of those privileges would suffice. If multiple rows are displayed for a stored procedure, then one privilege from each row is required to execute the stored procedure.

If sp_proc_priv is invoked without *proc_name*, it returns privilege information for all procedures that require privileges. System procedures that do not require privileges are not included in the results.

**Note**

This procedure only lists privileges for a stored procedure that never fails the permission check for the procedure. There may be other privileges that pass the permission check to execute the procedure under certain conditions, but those privileges are not listed.

Privileges

None

Side effects

None

See also

- “Roles” [SQL Anywhere Server - Database Administration]  
- “sp_sys_priv_role_info system procedure” on page 1311  
- “sp_has_role system procedure” on page 1284  
- “sp_displayroles system procedure” on page 1276  
- “sp_objectpermission system procedure” on page 1292
Example

The following example returns the privileges required to run the `sa_table_fragmentation` system procedure. Since only one row is returned, any one of the privileges in the comma separated list is sufficient to run `sa_table_fragmentation`.

```
CALL sp_proc_priv( 'sa_table_fragmentation' );
```

<table>
<thead>
<tr>
<th>proc_name</th>
<th>privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>sa_table_fragmentation</td>
<td>MANAGE ANY STATISTICS, MONITOR</td>
</tr>
</tbody>
</table>

The following example returns the privileges for the `sa_install_feature` system procedure. Since multiple rows are returned, one privilege from each row is required to run `sa_install_feature`.

```
CALL sp_proc_priv( 'sa_install_feature' );
```

<table>
<thead>
<tr>
<th>proc_name</th>
<th>privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>sa_install_feature</td>
<td>SELECT ANY TABLE</td>
</tr>
<tr>
<td>sa_install_feature</td>
<td>MANAGE ANY SPATIAL OBJECT</td>
</tr>
<tr>
<td>sa_install_feature</td>
<td>MANAGE ANY OBJECT PRIVILEGE</td>
</tr>
<tr>
<td>sa_install_feature</td>
<td>CREATE ANY PROCEDURE, CREATE ANY OBJECT</td>
</tr>
</tbody>
</table>

**sp_remote_columns** system procedure

Produces a list of the columns in a remote table, and a description of their data types.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax

```
sp_remote_columns(
    @server_name
    , @table_name
    [, @table_owner
    [, @table_qualifier ] ]
)
```

Arguments

- **@server_name** Use this CHAR(128) parameter to specify a string containing the server name as specified by the CREATE SERVER statement.

- **@table_name** Use this CHAR(128) parameter to specify the name of the remote table.

- **@table_owner** Use this optional CHAR(128) parameter to specify the owner of `table_name`. The default is '%'.

1298Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
• **@table_qualifier**  Use this optional CHAR(128) parameter to specify the name of the database in which *table_name* is located. The default is ‘%’.

## Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>CHAR(128)</td>
<td>The database name.</td>
</tr>
<tr>
<td>owner</td>
<td>CHAR(128)</td>
<td>The database owner name.</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>The table name.</td>
</tr>
<tr>
<td>column_name</td>
<td>CHAR(128)</td>
<td>The name of a column.</td>
</tr>
<tr>
<td>domain_id</td>
<td>SMALLINT</td>
<td>An INTEGER which indicates the data type of the column.</td>
</tr>
<tr>
<td>width</td>
<td>INTEGER</td>
<td>The meaning of this column depends on the data type. For character types width represents the number of characters.</td>
</tr>
<tr>
<td>scale</td>
<td>SMALLINT</td>
<td>The meaning of this column depends on the data type. For NUMERIC data types scale is the number of digits after the decimal point.</td>
</tr>
<tr>
<td>nullable</td>
<td>SMALLINT</td>
<td>If NULL column values are allowed, the value is 1. Otherwise the value is 0.</td>
</tr>
<tr>
<td>base_type_str</td>
<td>CHAR(4096)</td>
<td>The annotated type string representing the physical type of the column.</td>
</tr>
</tbody>
</table>

## Remarks

If you are entering a CREATE EXISTING TABLE statement and you are specifying a column list, it may be helpful to get a list of the columns that are available on a remote table. `sp_remote_columns` produces a list of the columns on a remote table and a description of their data types. If you specify a database, you must either specify an owner or provide the value NULL.

## Privileges

None

## Side effects

None

## See also

- “Remote data access” [SQL Anywhere Server - SQL Usage]
- “Server classes for remote data access” [SQL Anywhere Server - SQL Usage]
- “CREATE SERVER statement” on page 657
Standards and compatibility

- **Sybase**  Supported by Open Client/Open Server.

Example

The following example returns information about the columns in the ULProduct table in the remote SQL Anywhere database server named RemoteSA. The table owner is DBA.

```sql
CALL sp_remote_columns( 'RemoteSA', 'ULProduct', 'DBA', null );
```

The following example returns information about the columns in the SYSOBJECTS table in the Adaptive Server Enterprise database Production using the remote server named RemoteASE. The table owner is unspecified.

```sql
CALL sp_remote_columns( 'RemoteASE', 'sysobjects', null, 'Production' );
```

The following example returns formation about the columns in the Customers table in the Microsoft Access database `c:\users\me\documents\MyAccesDB.accdb` using the remote server MyAccessDB. The Access database does not have a table owner so NULL is specified.

```sql
CALL sp_remote_columns( 'MyAccessDB', 'Customers', null, 'c:\users\me\documents\MyAccesDB.accdb' );
```

sp_remote_exported_keys system procedure

Provides information about tables with foreign keys on a specified primary table.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax

```sql
sp_remote_exported_keys(
  @server_name
  , @table_name
  [ , @table_owner
  [ , @table_qualifier ] ]
)
```

Arguments

- **@server_name**  Use this CHAR(128) parameter to specify the server the primary table is located on.

- **@table_name**  Use this CHAR(128) parameter to specify the table containing the primary key.

- **@table_owner**  Use this optional CHAR(128) parameter to specify the primary table's owner. The default is '%'.

- **@table_qualifier**  Use this optional CHAR(128) parameter to specify the database containing the primary table. The default is '%'.


Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pk_database</td>
<td>CHAR(128)</td>
<td>The database containing the primary key table.</td>
</tr>
<tr>
<td>pk_owner</td>
<td>CHAR(128)</td>
<td>The owner of the primary key table.</td>
</tr>
<tr>
<td>pk_table</td>
<td>CHAR(128)</td>
<td>The primary key table.</td>
</tr>
<tr>
<td>pk_column</td>
<td>CHAR(128)</td>
<td>The name of the primary key column.</td>
</tr>
<tr>
<td>fk_database</td>
<td>CHAR(128)</td>
<td>The database containing the foreign key table.</td>
</tr>
<tr>
<td>fk_owner</td>
<td>CHAR(128)</td>
<td>The foreign key table's owner.</td>
</tr>
<tr>
<td>fk_table</td>
<td>CHAR(128)</td>
<td>The foreign key table.</td>
</tr>
<tr>
<td>fk_column</td>
<td>CHAR(128)</td>
<td>The name of the foreign key column.</td>
</tr>
<tr>
<td>key_seq</td>
<td>SMALLINT</td>
<td>The key sequence number.</td>
</tr>
<tr>
<td>fk_name</td>
<td>CHAR(128)</td>
<td>The foreign key name.</td>
</tr>
<tr>
<td>pk_name</td>
<td>CHAR(128)</td>
<td>The primary key name.</td>
</tr>
</tbody>
</table>

Remarks

This procedure provides information about the remote tables that have a foreign key on a particular primary table. The result set for the sp_remote_exported_keys system procedure includes the database, owner, table, column, and name for both the primary and the foreign key, and the foreign key sequence for the foreign key columns. The result set may vary because of the underlying ODBC and JDBC calls, but information about the table and column for a foreign key is always returned.

Privileges

None

Side effects

None

See also

- “CREATE SERVER statement” on page 657
- “Foreign keys” [SQL Anywhere Server - SQL Usage]

Example

This example returns information about the foreign key relationships in the ULEmployee table on the remote server named RemoteSA:

```
CALL sp_remote_exported_keys( 'RemoteSA', 'ULEmployee', 'DBA' );
```
sp_remote_imported_keys system procedure

Provides information about remote tables with primary keys that correspond to a specified foreign key.

The server must be defined with the CREATE SERVER statement to use this system procedure.

Syntax

```
sp_remote_imported_keys(
    @server_name
    , @table_name
    [, @table_owner
    [, @table_qualifier ] ]
)
```

Arguments

- **@server_name** Use this CHAR(128) parameter to specify the server the foreign key table is located on. A value is required for this parameter.

- **@table_name** Use this CHAR(128) parameter to specify the table containing the foreign key. A value is required for this parameter.

- **@table_owner** Use this optional CHAR(128) parameter to specify the foreign key table's owner. The default is '%'.

- **@table_qualifier** Use this optional CHAR(128) parameter to specify the database containing the foreign key table. The default is '%'.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pk_database</td>
<td>CHAR(128)</td>
<td>The database containing the primary key table.</td>
</tr>
<tr>
<td>pk_owner</td>
<td>CHAR(128)</td>
<td>The owner of the primary key table.</td>
</tr>
<tr>
<td>pk_table</td>
<td>CHAR(128)</td>
<td>The primary key table.</td>
</tr>
<tr>
<td>pk_column</td>
<td>CHAR(128)</td>
<td>The name of the primary key column.</td>
</tr>
<tr>
<td>fk_database</td>
<td>CHAR(128)</td>
<td>The database containing the foreign key table.</td>
</tr>
<tr>
<td>fk_owner</td>
<td>CHAR(128)</td>
<td>The foreign key table's owner.</td>
</tr>
<tr>
<td>fk_table</td>
<td>CHAR(128)</td>
<td>The foreign key table.</td>
</tr>
<tr>
<td>fk_column</td>
<td>CHAR(128)</td>
<td>The name of the foreign key column.</td>
</tr>
<tr>
<td>key_seq</td>
<td>SMALLINT</td>
<td>The key sequence number.</td>
</tr>
<tr>
<td>fk_name</td>
<td>CHAR(128)</td>
<td>The foreign key name.</td>
</tr>
</tbody>
</table>
### Remarks
Foreign keys reference a row in a separate table that contains the corresponding primary key. This procedure allows you to obtain a list of the remote tables with primary keys that correspond to a particular foreign table. The `sp_remote_imported_keys` result set includes the database, owner, table, column, and name for both the primary and the foreign key, and the foreign key sequence for the foreign key columns. The result set may vary because of the underlying ODBC and JDBC calls, but information about the table and column for a primary key is always returned.

### Privileges
None

### Side effects
None

### See also
- “CREATE SERVER statement” on page 657
- “Foreign keys” [SQL Anywhere Server - SQL Usage]

### Example
The following example returns the tables with primary keys that correspond to a foreign key on the ULOrder table on the remote server named RemoteSA:

```sql
CALL sp_remote_imported_keys( 'RemoteSA', 'ULOrder', 'DBA' );
```

### sp_remote_pcols system procedure
-produces a list of the columns in a remote table, and a description of their data types.

The server must be defined with the CREATE SERVER statement to use this system procedure.

#### Syntax
```sql
sp_remote_pcols(
    @server_name
    , @sp_name
    [, @sp_owner
    [, @sp_qualifier ] ]
)
```

#### Arguments
- **@server_name** Use this CHAR(128) parameter to specify a string containing the server name as specified by the CREATE SERVER statement.
● **@sp_name**  Use this CHAR(128) parameter to specify the name of the remote table.

● **@sp_owner**  Use this optional CHAR(128) parameter to specify the owner of *sp_name*. The default is '%'.

● **@sp_qualifier**  Use this optional CHAR(128) parameter to specify the name of the database in which *sp_name* is located. The default is '%'.

### Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>CHAR(128)</td>
<td>The database name.</td>
</tr>
<tr>
<td>owner</td>
<td>CHAR(128)</td>
<td>The database owner name.</td>
</tr>
<tr>
<td>proc_name</td>
<td>CHAR(128)</td>
<td>The stored procedure name.</td>
</tr>
<tr>
<td>parm_name</td>
<td>CHAR(128)</td>
<td>The name of the parameter or result set column.</td>
</tr>
<tr>
<td>parm_mode</td>
<td>CHAR(10)</td>
<td>The mode of the parameter or result set column (IN, OUT, INOUT, RESULT).</td>
</tr>
<tr>
<td>domain_id</td>
<td>SMALLINT</td>
<td>An INTEGER which indicates the data type of the parameter or result set column.</td>
</tr>
<tr>
<td>width</td>
<td>INTEGER</td>
<td>The meaning of this column depends on the data type. For character types width represents the number of characters.</td>
</tr>
<tr>
<td>scale</td>
<td>SMALLINT</td>
<td>The meaning of this column depends on the data type. For NUMERIC data types scale is the number of digits after the decimal point.</td>
</tr>
<tr>
<td>nullable</td>
<td>SMALLINT</td>
<td>If NULL parameter values are allowed, the value is 1. Otherwise the value is 0.</td>
</tr>
</tbody>
</table>

### Remarks

If you are entering a CREATE PROCEDURE ... AT statement and you are specifying a parameter list or want information on any result set that may be returned, it may be helpful to get a list of the parameters and result set columns that are available for a remote stored procedure. *sp_remote_pcols* produces a list of the parameters and result set columns of a remote stored procedure and a description of their data types. If you specify a database, you must either specify an owner or provide the value NULL.

### Privileges

None

### Side effects

None
See also

- “Remote data access” [SQL Anywhere Server - SQL Usage]
- “Server classes for remote data access” [SQL Anywhere Server - SQL Usage]
- “CREATE SERVER statement” on page 657

Standards and compatibility

- **Sybase** Supported by Open Client/Open Server.

Example

The following example returns information about the parameters and result set columns of the ULOrderDownload stored procedure in the remote SQL Anywhere database server named RemoteSA. The stored procedure owner is DBA.

```sql
CALL sp_remote_pcols('RemoteSA', 'ULOrderDownload', 'DBA');
```

The following example returns information about the parameters and result set columns of the col_name stored procedure in the Adaptive Server Enterprise database Production using the remote server named RemoteASE. The remote procedure owner is unspecified.

```sql
CALL sp_remote_pcols( 'RemoteASE', 'col_name', null, 'Production' );
```

**sp_remote_primary_keys system procedure**

Provides primary key information about remote tables using remote data access.

Syntax

```sql
sp_remote_primary_keys(
    @server_name
    , @table_name
    [, @table_owner
    [, @table_qualifier ] ]
)
```

Arguments

- **@server_name** Use this CHAR(128) parameter to specify the remote server name.
- **@table_name** Use this CHAR(128) parameter to specify the name of the remote table.
- **@table_owner** Use this optional CHAR(128) parameter to specify the owner of the remote table. The default is ‘%’.
- **@table_qualifier** Use this optional CHAR(128) parameter to specify the name of the remote database. The default is ‘%’.
### Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>CHAR(128)</td>
<td>The name of the remote database.</td>
</tr>
<tr>
<td>owner</td>
<td>CHAR(128)</td>
<td>The owner of the table.</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>column_name</td>
<td>CHAR(128)</td>
<td>The name of the primary key column.</td>
</tr>
<tr>
<td>key_seq</td>
<td>SMALLINT</td>
<td>The primary key sequence number.</td>
</tr>
<tr>
<td>pk_name</td>
<td>CHAR(128)</td>
<td>The primary key name.</td>
</tr>
</tbody>
</table>

### Remarks

This system procedure provides primary key information about remote tables using remote data access. Because of differences in the underlying ODBC calls, the information returned differs slightly from the catalog/database value depending upon the remote data access class that is specified for the server.

### Privileges

None

### Standards and compatibility

- **Sybase**  
  Supported by Open Client/Open Server.

### Side effects

None

### See also

- “Remote data access” [SQL Anywhere Server - SQL Usage]
- “Server classes for remote data access” [SQL Anywhere Server - SQL Usage]
- “CREATE SERVER statement” on page 657

### Examples

The following example returns information about the primary keys in tables owned by DBA in a SQL Anywhere remote server named RemoteSA.

```sql
CALL sp_remote_primary_keys( 'RemoteSA', null, 'DBA' );
```

To get a list of the primary keys in all the tables owned by Fred in the production database in an Adaptive Server Enterprise server named RemoteASE:

```sql
CALL sp_remote_primary_keys( 'RemoteASE', null, 'Fred', 'production' );
```
**sp_remote_procedures system procedure**

Returns a list of the procedures on a remote server.

**Syntax**

```sql
sp_remote_procedures(
    @server_name
    [, @sp_name
    [, @sp_owner
    [, @sp_qualifier ] ] ]
)
```

**Arguments**

- **@server_name**  Use this CHAR(128) parameter to specify the remote server name.

- **@sp_name**  Use this optional CHAR(128) parameter to specify the remote stored procedure name. The default is '%'.

- **@sp_owner**  Use this optional CHAR(128) parameter to specify the owner of the remote stored procedure. The default is '%'.

- **@sp_qualifier**  Use this optional CHAR(128) parameter to specify the database in which `sp_name` is located. The default is '%'.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>CHAR(128)</td>
<td>The name of the remote database.</td>
</tr>
<tr>
<td>owner</td>
<td>CHAR(128)</td>
<td>The name of the stored procedure owner.</td>
</tr>
<tr>
<td>proc_name</td>
<td>CHAR(128)</td>
<td>The name of the stored procedure.</td>
</tr>
</tbody>
</table>

**Remarks**

The server must be defined with the CREATE SERVER statement to use this system procedure.

It may be helpful when you are configuring your database server to get a list of the remote stored procedures available on a particular server. This procedure returns a list of the stored procedures on a server.

The procedure accepts four parameters. If a stored procedure, owner, or database name is given, the list of stored procedures will be limited to only those that match the arguments.

**Privileges**

None

**Side effects**

None
See also

- “Remote data access” [SQL Anywhere Server - SQL Usage]
- “Server classes for remote data access” [SQL Anywhere Server - SQL Usage]
- “CREATE PROCEDURE statement” on page 639
- “CREATE SERVER statement” on page 657

Standards and compatibility

- Sybase Supported by Open Client/Open Server.

Examples

The following example returns information about the stored procedures owned by DBA in a SQL Anywhere remote server named RemoteSA:

```sql
CALL sp_remote_procedures( 'RemoteSA', null, 'DBA' );
```

The following example returns information about the stored procedures owned by Fred in the production database in an Adaptive Server Enterprise remote server named RemoteASE:

```sql
CALL sp_remote_procedures( RemoteASE', null, 'Fred', 'production' );
```

**sp_remote_tables system procedure**

Returns a list of the tables on a server.

Syntax

```sql
sp_remote_tables(
  @server_name
  [, @table_name
  [, @table_owner
  [, @table_qualifier
  [, @with_table_type ] ] ] ] ] ]
)
```

Arguments

- **@server_name** Use this CHAR(128) parameter to specify the remote server name.

- **@table_name** Use this optional CHAR(128) parameter to specify the name of the remote table. The default is '%'.

- **@table_owner** Use this optional CHAR(128) parameter to specify the owner of the remote table. The default is '%'.

- **@table_qualifier** Use this optional CHAR(128) parameter to specify the database in which `table_name` is located. The default is '%'.

- **@with_table_type** Use this optional BIT parameter to specify the inclusion of remote table types. The default is 0. Specify 1 if you want the result set to include a column that lists table types or specify 0 if you do not.
Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>CHAR(128)</td>
<td>The name of the remote database.</td>
</tr>
<tr>
<td>owner</td>
<td>CHAR(128)</td>
<td>The name of the table owner.</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>table_type</td>
<td>CHAR(128)</td>
<td>Specifies the table type. The value depends on the type of remote server. For example, TABLE, VIEW, SYS, and GBL TEMP are possible values.</td>
</tr>
</tbody>
</table>

Remarks

The server must be defined with the CREATE SERVER statement to use this system procedure.

It may be helpful when you are configuring your database server to get a list of the remote tables available on a particular server. This procedure returns a list of the tables on a server.

The procedure accepts five parameters. If a table, owner, or database name is given, the list of tables will be limited to only those that match the arguments.

Privileges

None

Side effects

None

Standards and compatibility

- **Sybase**  
  Supported by Open Client/Open Server.

See also

- “Remote data access” [SQL Anywhere Server - SQL Usage]
- “Server classes for remote data access” [SQL Anywhere Server - SQL Usage]
- “CREATE SERVER statement” on page 657

Examples

The following example returns information about the tables owned by DBA in a SQL Anywhere remote server named RemoteSA.

```sql
CALL sp_remote_tables( 'RemoteSA', null, 'DBA' );
```

To get a list of all the tables owned by Fred in the production database in an Adaptive Server Enterprise server named RemoteASE:

```sql
CALL sp_remote_tables( 'RemoteASE', null, 'Fred', 'production' );
```
To get a list of all the Microsoft Excel worksheets available from an ODBC data source referenced by a server named RemoteExcel:

```
CALL sp_remote_tables( 'RemoteExcel' );
```

---

**sp_servercaps system procedure**

Displays information about a remote server's capabilities.

**Syntax**

```
sp_servercaps( @server_name )
```

**Arguments**

- `@server_name` Use this CHAR(128) parameter to specify a server defined with the CREATE SERVER statement. `@server_name` is the same server name used in the CREATE SERVER statement.

**Results**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capid</td>
<td>INTEGER</td>
<td>The capability identifier.</td>
</tr>
<tr>
<td>capname</td>
<td>CHAR(128)</td>
<td>The name of the capability.</td>
</tr>
<tr>
<td>capvalue</td>
<td>CHAR(128)</td>
<td>The setting of the capability, usually T (true) or F (false).</td>
</tr>
</tbody>
</table>

**Remarks**

The server must be defined with the CREATE SERVER statement to use this system procedure.

This procedure displays information about a remote server's capabilities. The capability information is used to determine how much of a SQL statement can be forwarded to a remote server. The ISYSCAPABILITY system table, which lists the server capabilities, is not populated until a connection is made to the first remote server.

**Standards and compatibility**

- **Sybase** Supported by Open Client/Open Server.

**Privileges**

None

**Side effects**

None
Example
To display information about the remote server RemoteSA:

\[
\text{CALL sp_servercaps( 'RemoteSA' );}
\]

sp_sys_priv_role_info system procedure

Returns the one-to-one mapping between system privileges and system roles.

Syntax

\[
\text{sp_sys_priv_role_info( )}
\]

Result set

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_priv_name</td>
<td>CHAR(128)</td>
<td>The name of the system privilege.</td>
</tr>
<tr>
<td>sys_priv_role_name</td>
<td>CHAR(128)</td>
<td>The name of the corresponding system role.</td>
</tr>
<tr>
<td>sys_priv_id</td>
<td>UNSIGNED INTEGER</td>
<td>The ID number for the system role.</td>
</tr>
</tbody>
</table>

Remarks

sp_sys_priv_role_info reports the complete one-to-one mapping between system privileges and system roles and role IDs. This procedure returns a row for each system privilege.

Designed mainly for internal use, this procedure is used by utilities that need to map a granted system role to the corresponding system privilege and vice versa.

Privileges

None

See also

- “Roles” [SQL Anywhere Server - Database Administration]
- “sp_has_role system procedure” on page 1284
- “sp_proc_priv system procedure” on page 1297
- “sp_displayroles system procedure” on page 1276
- “sp_objectpermission system procedure” on page 1292
Standards and compatibility

- SQL/2008 Vendor extension.

Example

The following statement returns the mapping of system privileges to their corresponding system role.

```
CALL sp_sys_priv_role_info();
```

<table>
<thead>
<tr>
<th>sys_priv_name</th>
<th>sys_priv_role_name</th>
<th>sys_priv_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALIDATE ANY OBJECT</td>
<td>SYS_VALIDATE_ANY_OBJECT_ROLE</td>
<td>2147483819</td>
</tr>
<tr>
<td>REORGANIZE ANY OBJECT</td>
<td>SYS_REORGANIZE_ANY_OBJECT_ROLE</td>
<td>2147483820</td>
</tr>
<tr>
<td>BACKUP DATABASE</td>
<td>SYS_BACKUP_DATABASE_ROLE</td>
<td>2147483821</td>
</tr>
<tr>
<td>MANAGE ANY EVENT</td>
<td>SYS_MANAGE_ANY_EVENT_ROLE</td>
<td>2147483822</td>
</tr>
<tr>
<td>ALTER DATABASE</td>
<td>SYS_ALTER_DATABASE_ROLE</td>
<td>2147483823</td>
</tr>
<tr>
<td>SERVER OPERATOR</td>
<td>SYS_SERVER_OPERATOR_ROLE</td>
<td>2147483824</td>
</tr>
<tr>
<td>UPGRADE ROLE</td>
<td>SYS_UPGRADE_ROLE_ROLE</td>
<td>2147483825</td>
</tr>
<tr>
<td>MANAGE ANY LDAP SERVER</td>
<td>SYS_MANAGE_ANY_LDAP_SERVER_ROLE</td>
<td>2147483827</td>
</tr>
<tr>
<td>MANAGE CERTIFICATES</td>
<td>SYS_MANAGE_CERTIFICATES_ROLE</td>
<td>2147483828</td>
</tr>
<tr>
<td>CREATE ANY SEQUENCE</td>
<td>SYS_CREATE_ANY_SEQUENCE_ROLE</td>
<td>2147483829</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**sp_trace_event_fields** system procedure

Returns information about the fields of the specified trace event.

Syntax

```
sp_trace_event_fields( [ event_name ] )
```

Arguments

- **event_name** Use this optional CHAR(256) parameter to specify the trace event name. The default is NULL.
### Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_name</td>
<td>CHAR(256)</td>
<td>Returns the name of the trace event. System trace event names have the prefix SYS_.</td>
</tr>
<tr>
<td>field_name</td>
<td>CHAR(256)</td>
<td>Returns the field name.</td>
</tr>
<tr>
<td>field_id</td>
<td>INTEGER</td>
<td>Returns the order of the field in the trace event.</td>
</tr>
<tr>
<td>field_domain</td>
<td>INTEGER</td>
<td>Returns the domain ID of the field. Foreign key to SYSDOMAIN.domain_id.</td>
</tr>
<tr>
<td>field_description</td>
<td>LONG VARCHAR</td>
<td>Returns a description of the field content.</td>
</tr>
</tbody>
</table>

### Remarks

If `event_name` is NULL, this procedure returns the fields for all trace events.

### Privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

### Side effects

None

### See also

- “Event tracing” [*SQL Anywhere Server - Database Administration*]
- “CREATE TEMPORARY TRACE EVENT statement” on page 705
- “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
- “ALTER TRACE EVENT SESSION statement” on page 506
- “DROP TRACE EVENT statement” on page 783
- “DROP TRACE EVENT SESSION statement” on page 784
- “NOTIFY TRACE EVENT statement” on page 903
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_sessions system procedure” on page 1317
- “sp_trace_event_session_events system procedure” on page 1314
- “sp_trace_event_session_targets system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [*SQL Anywhere Server - Database Administration*]
- “System events” [*SQL Anywhere Server - Database Administration*]

### Example

The following statement returns information about the fields for all trace event sessions in the database:

```sql
SELECT * FROM sp_trace_event_fields( );
```
sp_trace_event_session_events system procedure

Lists the trace events that are part of a specific trace event session.

Syntax

```
sp_trace_event_session_events([ session_name ])
```

Parameters

- `session_name` Use this optional CHAR(256) parameter to specify the name of the trace event session. The default is NULL. If a session name is not specified or is NULL, information is returned for all trace event sessions.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_name</td>
<td>CHAR(256)</td>
<td>Returns the name of the trace event session.</td>
</tr>
<tr>
<td>event_name</td>
<td>CHAR(256)</td>
<td>Returns the trace event name.</td>
</tr>
</tbody>
</table>

Remarks

Use this system procedure to determine which trace events (both system defined and user defined) are being logged for a trace event session.

Privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side effects

None

See also

- “Event tracing” [SQL Anywhere Server - Database Administration]
- “CREATE TEMPORARY TRACE EVENT statement” on page 705
- “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
- “ALTER TRACE EVENT SESSION statement” on page 506
- “DROP TRACE EVENT statement” on page 783
- “DROP TRACE EVENT SESSION statement” on page 784
- “NOTIFY TRACE EVENT statement” on page 903
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_fields system procedure” on page 1312
- “sp_trace_event_sessions system procedure” on page 1317
- “sp_trace_event_session_targets system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]
Example

This statement returns information about the events that are part of all the event tracing sessions for the current database:

```
SELECT * FROM sp_trace_event_session_events();
```

**sp_trace_event_session_target_options system procedure**

Lists the target options for a trace event session.

**Syntax**

```
sp_trace_event_session_target_options( [ session_name ] )
```

**Arguments**

- **session_name** Use this optional CHAR(256) parameter to specify the name of the trace event session. The default is NULL. If a session name is not specified or is NULL, information is returned for all trace event sessions.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_name</td>
<td>CHAR(256)</td>
<td>Returns the session name.</td>
</tr>
<tr>
<td>target_type</td>
<td>CHAR(256)</td>
<td>Returns the target type (such as, FILE).</td>
</tr>
<tr>
<td>option_name</td>
<td>CHAR(256)</td>
<td>Returns the option name.</td>
</tr>
<tr>
<td>option_value</td>
<td>LONG VARCHAR</td>
<td>Returns the option value.</td>
</tr>
</tbody>
</table>

**Remarks**

This procedure returns information about the options for one or all trace event sessions for the current database.

**Privileges**

You must have the MANAGE ANY TRACE SESSION system privilege.

**Side effects**

None
Example

The following statement returns information about target options for all trace event sessions in the database:

```
SELECT * FROM sp_trace_event_session_target_options( );
```

### sp_trace_event_session_targets system procedure

Lists the targets of a trace session.

**Syntax**

```
sp_trace_event_session_targets( [ session_name ] )
```

**Arguments**

- **session_name**  
  Use this optional CHAR(256) parameter to specify the name of the trace event session. The default is NULL. If a session name is not specified or is NULL, information is returned about all trace event sessions in the database.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_name</td>
<td>CHAR(256)</td>
<td>Returns the session name.</td>
</tr>
<tr>
<td>target_type</td>
<td>CHAR(256)</td>
<td>Returns the target type (for example, a file).</td>
</tr>
</tbody>
</table>

**Remarks**

A target is the location where the trace event session information is being logged (such as a file or database server message log).
Privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side effects

None

See also

● “Event tracing” [SQL Anywhere Server - Database Administration]
● “CREATE TEMPORARY TRACE EVENT statement” on page 705
● “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
● “ALTER TRACE EVENT SESSION statement” on page 506
● “DROP TRACE EVENT statement” on page 783
● “DROP TRACE EVENT SESSION statement” on page 784
● “NOTIFY TRACE EVENT statement” on page 903
● “sp_trace_events system procedure” on page 1318
● “sp_trace_event_fields system procedure” on page 1312
● “sp_trace_event_sessions system procedure” on page 1317
● “sp_trace_event_session_events system procedure” on page 1314
● “sp_trace_event_session_target_options system procedure” on page 1315
● “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
● “System events” [SQL Anywhere Server - Database Administration]

Example

The following statement returns information about all trace event sessions in the database:

```
SELECT * FROM sp_trace_event_session_targets();
```

sp_trace_event_sessions system procedure

Returns a list of the trace event sessions that are defined for the database.

Syntax

```
sp_trace_event_sessions( [ session_name ] )
```

Parameters

● session_name  Use this CHAR(256) parameter to specify the trace event session you want to get information about. If a session name is not specified, information is returned about all trace event sessions in the database.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session_name</td>
<td>CHAR(256)</td>
<td>Returns the name of the session.</td>
</tr>
</tbody>
</table>
### Column name | Data type | Description
---|---|---
description | LONG VARCHAR | Returns a description of the session.
started | BIT | Returns 1 if the session has started and 0 otherwise.
is_temporary | BIT | Returns 1 if the trace session is temporary and 0 otherwise.

### Remarks
Start and stop trace event sessions manually by using the STATE clause of the ALTER TRACE EVENT SESSION statement.

### Privileges
You must have the MANAGE ANY TRACE SESSION system privilege.

### Side effects
None

### See also
- “Event tracing” [SQL Anywhere Server - Database Administration]
- “CREATE TEMPORARY TRACE EVENT statement” on page 705
- “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
- “ALTER TRACE EVENT SESSION statement” on page 506
- “DROP TRACE EVENT statement” on page 783
- “DROP TRACE EVENT SESSION statement” on page 784
- “NOTIFY TRACE EVENT statement” on page 903
- “sp_trace_events system procedure” on page 1318
- “sp_trace_event_fields system procedure” on page 1312
- “sp_trace_event_session_events system procedure” on page 1314
- “sp_trace_event_session_targets system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]

### Example
The following statement returns a list of trace event sessions for the current database:

```sql
SELECT * FROM sp_trace_event_sessions();
```

### sp_trace_events system procedure
Returns information about the trace events in the database.

**Syntax**

```
sp_trace_events([ event_name ])
```
Arguments

- **event_name**
  Use this optional CHAR(256) parameter to specify the trace event name. The default is NULL.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_name</td>
<td>CHAR(256)</td>
<td>Returns the name of the trace event. System trace event names have the prefix SYS_.</td>
</tr>
<tr>
<td>description</td>
<td>LONG VARCHAR</td>
<td>Returns a description of what the trace event captures.</td>
</tr>
<tr>
<td>severity</td>
<td>TINYINT</td>
<td>Returns the severity level of the trace event.</td>
</tr>
<tr>
<td>is_system</td>
<td>BIT</td>
<td>Returns 1 for system trace events and 0 otherwise.</td>
</tr>
<tr>
<td>is_temporary</td>
<td>BIT</td>
<td>Returns 1 for temporary trace events and 0 otherwise.</td>
</tr>
</tbody>
</table>

Remarks

This procedure returns information about both system-defined and user-defined trace events in the database. It returns the details for the specified trace event or for all trace events if `event_name` is not supplied or is NULL.

<table>
<thead>
<tr>
<th>Level</th>
<th>Severity value range</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALWAYS</td>
<td>0</td>
</tr>
<tr>
<td>CRITICAL</td>
<td>1-50</td>
</tr>
<tr>
<td>ERROR</td>
<td>51-100</td>
</tr>
<tr>
<td>WARNING</td>
<td>101-150</td>
</tr>
<tr>
<td>INFORMATION</td>
<td>151-200</td>
</tr>
<tr>
<td>DEBUG</td>
<td>201-255</td>
</tr>
</tbody>
</table>

Privileges

You must have the MANAGE ANY TRACE SESSION system privilege.

Side effects

None
See also

- “Event tracing” [SQL Anywhere Server - Database Administration]
- “CREATE TEMPORARY TRACE EVENT statement” on page 705
- “CREATE TEMPORARY TRACE EVENT SESSION statement” on page 707
- “ALTER TRACE EVENT SESSION statement” on page 506
- “DROP TRACE EVENT statement” on page 783
- “DROP TRACE EVENT SESSION statement” on page 784
- “NOTIFY TRACE EVENT statement” on page 903
- “sp_trace_event_fields system procedure” on page 1312
- “sp_trace_event_sessions system procedure” on page 1317
- “sp_trace_event_session_events system procedure” on page 1314
- “sp_trace_event_session_targets system procedure” on page 1316
- “sp_trace_event_session_target_options system procedure” on page 1315
- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]
- “System events” [SQL Anywhere Server - Database Administration]

Example

The following statement returns a list of user-defined trace events in the database:

```sql
SELECT * FROM sp_trace_events();
WHERE is_system = 0;
```

### sp_tsql_environment system procedure

Sets connection options when users connect from jConnect or Open Client applications.

#### Syntax

```sql
sp_tsql_environment();
```

#### Remarks

The `sp_login_environment` procedure is the default procedure specified by the `login_procedure` database option. For each new connection, the procedure specified by `login_procedure` is called. If the connection uses the TDS communications protocol (that is, if it is an Open Client or jConnect connection), then `sp_login_environment` in turn calls `sp_tsql_environment`.

This procedure sets database options so that they are compatible with default Adaptive Server Enterprise behavior.

To change the default behavior, create new procedures and alter your `login_procedure` option to point to these new procedures.

Below is the list of the options set by `sp_tsql_environment` procedure:

```sql
if db_property( 'IQStore' ) = 'Off' then
  -- SQL Anywhere datastore
  SET TEMPORARY OPTION close_on_endtrans='OFF';
end if;
SET TEMPORARY OPTION ansinull='OFF';
```
SET TEMPORARY OPTION tsql_variables='ON';
SET TEMPORARY OPTION ansi_blanks='ON';
SET TEMPORARY OPTION chained='OFF';
SET TEMPORARY OPTION quoted_identifier='OFF';
SET TEMPORARY OPTION allow_nulls_by_default='OFF';
SET TEMPORARY OPTION on_tsql_error='CONTINUE';
SET TEMPORARY OPTION isolation_level='1';
SET TEMPORARY OPTION date_format='YYYY-MM-DD';
SET TEMPORARY OPTION timestamp_format='YYYY-MM-DD HH:NN:SS.SSS';
SET TEMPORARY OPTION time_format='HH:NN:SS.SSS';
SET TEMPORARY OPTION date_order='MDY';
SET TEMPORARY OPTION escape_character='OFF';

Privileges
None

Side effects
None

See also
● “sp_login_environment system procedure” on page 1289
● “login_procedure option” [SQL Anywhere Server - Database Administration]

Example
The example below calls the sp_tsql_environment procedure:

CALL sp_tsql_environment();

sp_use_secure_feature_key system procedure
Enable the features included in the specified secure feature key for the current connection.

Syntax
sp_use_secure_feature_key(
    name
, auth_key
)

Arguments
● name    The VARCHAR (128) name of the secure feature key to be enabled.

● auth_key The CHAR (128) case-sensitive authorization key for the secure feature key being enabled. The authorization key must be at least six characters.

Remarks
This procedure enables the secure features that are included in the specified secure feature key for the current connection only.
Privileges

None.

Side effects

None.

See also

- “Creating secure feature keys” [SQL Anywhere Server - Database Administration]
- “sp_alter_secure_feature_key system procedure” on page 1268
- “sp_create_secure_feature_key system procedure” on page 1273
- “sp_drop_secure_feature_key system procedure” on page 1279
- “sp_list_secure_feature_keys system procedure” on page 1288

Example

In order for the following examples to work, the server must be started with the option: 

```
--sk

securefkey.
```

The following example enables the SYSTEM secure feature set which includes MANAGE_KEYS for the current connection.

```
CALL sp_use_secure_feature_key( 'system', 'securefkey' );
```

The following example enables the SYSTEM secure feature set which includes MANAGE_KEYS, creates a new secure feature key called MYSET with case-sensitive authorization key securemyset, and then uses the new secure feature key:

```
CALL sp_use_secure_feature_key( 'system', 'securefkey' );
CALL sp_create_secure_feature_key( 'myset', 'securemyset', 'local,remote' );
CALL sp_use_secure_feature_key( 'myset', 'securemyset' );
```

After switching to the new secure feature key, the current connection is no longer using the SYSTEM secure feature so does not have access to MANAGE_KEYS.

**st_geometry_dump system procedure**

Disassembles a geometry into its lowest level component geometries.

Syntax

```
st_geometry_dump( 
    geometry 
    [, "options" ]
)
```

Arguments

- **geometry** The ST_Geometry value of the geometry to be disassembled.

- **"options"** A optional VARCHAR(255) string of parameters and values, separated by semicolons, you can use to configure the output of the procedure. The default is NULL.
The following table lists the parameters that can be specified:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
<th>Allowed values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>Original</td>
<td>Original, Internal, or Mixed</td>
<td>The format to return the geometry in. Specifying Original returns the geometry in its original format. Specifying Internal returns the geometry in its normalized format. Specifying Mixed returns whatever stored formats are available, one row per format.</td>
</tr>
<tr>
<td>ExpandPoints</td>
<td>Yes</td>
<td>Yes, No</td>
<td>By default, when disassembling a geometry containing points (such as ST_LineString or ST_MultiPoint), the st_geometry_dump system procedure outputs the constituent points to separate rows. Set ExpandPoints to No if you do not want these extra rows to be generated.</td>
</tr>
<tr>
<td>MaxDepth</td>
<td>-1</td>
<td>-1, any number greater or equal to zero</td>
<td>By default, st_geometry_dump system procedure continues to disassembles an object hierarchy until it reaches the leaf objects. The MaxDepth parameter can be set to limit the number of levels in the hierarchy the geometry is disassembled. With a value of 0, only the root geometry is returned. With a value of 1, the geometry and its immediate children are returned, and so on.</td>
</tr>
<tr>
<td>SetGeom</td>
<td>Yes</td>
<td>Yes, No</td>
<td>The st_geometry_dump system procedure returns a column that is the ST_Geometry associated with an object in the original type hierarchy. If this column is not needed, the parameter SetGeom can be set to No to reduce the running time and output size of the procedure.</td>
</tr>
<tr>
<td>Validate</td>
<td>Basic</td>
<td>None, Basic, Full</td>
<td>By default, the st_geometry_dump system procedure applies the validation rules that the database server uses when loading geometries, and sets the Valid column of the result set to 1 if the object in the row matches these rules. The Validate parameter can be set to None to disable this checking, or it can be set to Full to also apply the additional checks performed by the ST_IsValid method. Full checking takes longer to perform.</td>
</tr>
</tbody>
</table>

Result set
The following table describes the results returned by the st_geometry_dump procedure:

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique ID for this row in the results.</td>
</tr>
<tr>
<td>Column</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parent_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the immediate parent of this object.</td>
</tr>
<tr>
<td>depth</td>
<td>INTEGER</td>
<td>The depth from the root object to the object associated with this row.</td>
</tr>
<tr>
<td>format</td>
<td>VARCHAR(128)</td>
<td>Whether the geometry is the original representation (Original) or the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>normalized representation (Internal).</td>
</tr>
<tr>
<td>valid</td>
<td>BIT</td>
<td>Whether the geometry is valid (1) according to the checking level</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specified by the Validate option.</td>
</tr>
<tr>
<td>geom_type</td>
<td>VARCHAR(128)</td>
<td>The geometry type, as returned by the ST_GeometryType.</td>
</tr>
<tr>
<td>geom</td>
<td>ST_Geometry</td>
<td>The geometry specification. If SetGeom parameter is set to No,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the geometry specification is not returned in the result set.</td>
</tr>
<tr>
<td>xmin</td>
<td>DOUBLE</td>
<td>The minimum x value for the geometry.</td>
</tr>
<tr>
<td>xmax</td>
<td>DOUBLE</td>
<td>The maximum x value for the geometry.</td>
</tr>
<tr>
<td>ymin</td>
<td>DOUBLE</td>
<td>The minimum y value for the geometry.</td>
</tr>
<tr>
<td>ymax</td>
<td>DOUBLE</td>
<td>The maximum y value for the geometry.</td>
</tr>
<tr>
<td>zmin</td>
<td>DOUBLE</td>
<td>The minimum z value for the geometry.</td>
</tr>
<tr>
<td>zmax</td>
<td>DOUBLE</td>
<td>The maximum z value for the geometry.</td>
</tr>
<tr>
<td>mmin</td>
<td>DOUBLE</td>
<td>The minimum m value for the geometry.</td>
</tr>
<tr>
<td>mmax</td>
<td>DOUBLE</td>
<td>The maximum m value for the geometry.</td>
</tr>
<tr>
<td>details</td>
<td>LONG VARCHAR</td>
<td>Any extra details about the geometry, including additional information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>about why the object is not valid.</td>
</tr>
</tbody>
</table>

**Remarks**

The `st_geometry_dump` system procedure disassembles a geometry hierarchy with one row for each of the objects in the hierarchy (including the root object). Each geometry in the hierarchy can be validated to find out if it is valid, and if not, why.

Some of the functionality of the `st_geometry_dump` system procedure can be matched by using type-specific methods such as `ST_GeometryN` or `ST_PointN`.

The `st_geometry_dump` system procedure can be used to correct invalid geometries.
Privileges
None

Side effects
None

See also
- “ST_IsValid method for type ST_Geometry” [SQL Anywhere Server - Spatial Data Support]
- “CREATE SPATIAL REFERENCE SYSTEM statement” on page 673
- “st_geometry_on_invalid option” [SQL Anywhere Server - Database Administration]
- “STORAGE FORMAT clause, CREATE SPATIAL REFERENCE SYSTEM statement” on page 679

Example
The following example disassembles the polygon, "Polygon ((0 0, 3 0, 3 3, 0 3, 0 0))", into its component geometries:

```
SELECT * FROM st_geometry_dump( 'Polygon ((0 0, 3 0, 3 3, 0 3, 0 0))', 'SetGeom=No' );
```

<table>
<thead>
<tr>
<th>id</th>
<th>parent_id</th>
<th>depth</th>
<th>format</th>
<th>valid</th>
<th>geom_type</th>
<th>geom</th>
<th>xmin</th>
<th>xmax</th>
<th>ymin</th>
<th>ymax</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Internal</td>
<td>1</td>
<td>ST_Polygon</td>
<td>Polygon</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Internal</td>
<td>1</td>
<td>ST_LineString</td>
<td>LineString</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>Internal</td>
<td>1</td>
<td>ST_Point</td>
<td>Point</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>Internal</td>
<td>1</td>
<td>ST_Point</td>
<td>Point</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>Internal</td>
<td>1</td>
<td>ST_Point</td>
<td>Point</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
The following example shows how the st_geometry_dump system procedure can be used to find the invalid points within a geometry. In this example, the linestring contains a point with longitude 1200. Because of this, the point and the linestring are both reported as invalid (valid=0) in the results.

```sql
SET TEMPORARY OPTION st_geometry_on_invalid='Ignore';
CREATE OR REPLACE VARIABLE @geo ST_Geometry;
SET @geo = new ST_LineString( 'LineString(1200 2, 80 10)', 4326 );
SELECT * FROM st_geometry_dump( @geo, 'SetGeom=No' );
```

Once invalid data has been identified, the st_geometry_dump system procedure can be used with other spatial methods to correct the invalid elements to assemble a valid geometry. The following example shows how an invalid point with longitude 1200 can be corrected to have longitude 120.0:

```sql
SELECT * FROM st_geometry_dump( @geo, 'SetGeom=No' );
```
SET TEMPORARY OPTION st_geometry_on_invalid='Ignore';
CREATE OR REPLACE VARIABLE @geo ST_Geometry;
SET @geo = new ST_LineString( 'LineString(1200 2, 80 10)', 4326 );

SELECT ST_LineString::ST_LineStringAggr(
    new ST_Point( IF xmax = 1200 then 120.0 ELSE xmax ENDIF,
    ymax, 4326 ) ORDER BY id )
FROM st_geometry_dump( @geo )
WHERE geom_type='ST_Point';

st_geometry_load_shapefile system procedure

Creates a table and loads an ESRI shapefile into it.

Syntax

st_geometry_load_shapefile(
  shp_filename
,   srid
,   table_name
[,   table_owner
[   ,   shp_encoding ] ]
)

Arguments

- **shp_filename** Use this VARCHAR(512) parameter to specify the location of the ESRI shapefile. The file name must have the extension .shp and must have associated .shx and .dbf files with the same base name located in the same directory. The path is relative to the database server, not the client application.

- **srid** Use this INTEGER parameter to specify the SRID to associate with the data from the shapefile. For the geometries to be meaningful, the SRID you specify should be appropriate for the geometries, even if it isn't identical to the SRID used in the shapefile.

- **table_name** Use this VARCHAR(128) column to specify the name of the table to create to hold the shapefile data.

- **table_owner** Use this optional VARCHAR(128) column to specify the owner for the new table that is created. The default owner is the current user.

- **shp_encoding** Use this optional VARCHAR(50) parameter to specify the encoding to use when reading the shapefile. The default encoding is ISO-8859-1.

Remarks

The st_geometry_load_shapefile system procedure first creates the table "table_owner","table_name" using the column information (names and types) found in the ESRI shapefile. st_geometry_load_shapefile then loads the data from the shapefile into the new table.

This procedure makes use of the sa_describe_shapefile system procedure, and the LOAD TABLE...FORMAT SHAPEFILE statement.
Privileges

If the -gl option is set to NONE, loading the data fails.

If the -gl option is set to DBA:

- If you are table_owner, you must have one of CREATE TABLE, CREATE ANY TABLE, or CREATE ANY OBJECT system privilege, and you must have one of ALTER ANY TABLE, ALTER ANY OBJECT, or LOAD ANY TABLE system privilege.

- If you are not table_owner, you must have one of CREATE ANY TABLE or CREATE ANY OBJECT system privilege, and you must have one of ALTER ANY TABLE, ALTER ANY OBJECT, or LOAD ANY TABLE system privilege.

If the -gl option is set to ALL:

- If you are table_owner, you must have one of CREATE TABLE, CREATE ANY TABLE, or CREATE ANY OBJECT system privilege.

- If you are not table_owner, you must have one of CREATE ANY TABLE or CREATE ANY OBJECT system privilege, and you must have one of ALTER ANY TABLE, ALTER ANY OBJECT, or LOAD ANY TABLE system privilege.

Side effects

If loading fails, the created table remains.

See also

- “sa_describe_shapefile system procedure” on page 1130
- “Lesson 3: Load the ESRI shapefile data” [SQL Anywhere Server - Spatial Data Support]
- “LOAD TABLE statement” on page 873
- “Support for ESRI shapefiles” [SQL Anywhere Server - Spatial Data Support]

Example

The following statement creates the table esri_load and loads it with the data from a fictitious shapefile c:\esri\shapefile.shp, associating the data with SRID 1000004326:

```sql
CALL st_geometry_load_shapefile( 'c:\esri\tgr36069trt00.shp', 1000004326, 'esri_load' );
```

xp_cmdshell system procedure

Carries out an operating system command from a procedure.

Syntax

```sql
xp_cmdshell(
    command
    [, redir_output | 'no_output'
)```
Arguments

- **command**  Use this VARCHAR(8000) parameter to specify a system command. The default is NULL.

- **redir_output**  Use this optional CHAR(254) parameter to specify whether to display output in a command window. The default behavior is to display output in a command window. If you specify 'no_output', output is not displayed in a command window. The default value is ''.

Returns

This function returns an INTEGER exit code.

Remarks

The second parameter affects only command line applications on Windows operating systems. For Unix, no command window appears, regardless of the setting for the second parameter.

For Windows Mobile, any commands executed are visible in the database server message log, regardless of the setting for the second parameter. The console shell `\windows\cmd.exe` is needed to run the procedure.

Privileges

You must have the SERVER OPERATOR system privilege.

See also

- “CALL statement” on page 530
- “Directory and file system procedures” on page 1088
- “General security tips” [SQL Anywhere Server - Database Administration]

Examples

The following statement lists the files in the current directory in the file `c:\temp.txt`:

```sql
CALL xp_cmdshell( 'dir > c:\temp.txt' );
```

The following statement carries out the same operation, but does so without displaying a Command window.

```sql
CALL xp_cmdshell( 'dir > c:\temp.txt', 'no_output' );
```

**xp_get_mail_error_code system procedure**

Returns information about the most recent SMTP or MAPI error.

**Syntax**

```
xp_get_mail_error_code( )
```
Returns
This function returns an INTEGER value representing the SMTP or MAPI error code.

Remarks
When the return value of a mail procedure (xp_startmail, xp_startsmtp, xp_sendmail, xp_stopmail, and xp_stopsmtp) is -1, use this function to retrieve the SMTP or MAPI error code.

When the return value of a mail procedure is 5, 6, or 7, use this function to retrieve the errno value for the most recent socket error.

If MAPI is being used, the value returned is the return code of the MAPI function. If SMTP is being used, the value returned is either an SMTP error code (and xp_get_mail_error_text returns the SMTP error text) or an errno value (and xp_get_mail_error_text returns an empty string).

Privileges
You must have the SEND EMAIL system privilege.

Side effects
None

See also
- “Return codes for MAPI and SMTP system procedures” on page 1086
- “xp_get_mail_error_text system procedure” on page 1331
- “xp_startmail system procedure” on page 1340
- “xp_startsmtp system procedure” on page 1341
- “xp_sendmail system procedure” on page 1336
- “xp_stopmail system procedure” on page 1343
- “xp_stopsmtp system procedure” on page 1344

Example
This example gets the most recent SMTP or MAPI error code.

```sql
SELECT xp_get_mail_error_code( )
```

This example uses SMTP to initiate the sending of a plain text message.

```sql
BEGIN
    DECLARE err_smtp INTEGER;
    DECLARE err_code INTEGER;
    DECLARE err_msg LONG VARCHAR;

    SELECT xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' ) INTO err_smtp;
    SELECT xp_get_mail_error_code( ), xp_get_mail_error_text( ) INTO err_code, err_msg;
    SELECT err_smtp, err_code, err_msg;
END;
```
xp_get_mail_error_text system procedure

Returns the most recent SMTP error or status message text.

Syntax

xp_get_mail_error_text()

Return value

This function returns a LONG VARCHAR value representing the SMTP or MAPI error or status message text. If no error text is available, an empty string or NULL is returned.

Remarks

Use this function to obtain the error or status message text for any of the mail procedures (xp_startmail, xp_startsmtp, xp_sendmail, xp_stopmail, and xp_stopsmtp).

Privileges

You must have the SEND EMAIL system privilege.

Side effects

None

See also

- “Return codes for MAPI and SMTP system procedures” on page 1086
- “xp_get_mail_error_code system procedure” on page 1329
- “xp_startmail system procedure” on page 1340
- “xp_startsmtp system procedure” on page 1341
- “xp_sendmail system procedure” on page 1336
- “xp_stopmail system procedure” on page 1343
- “xp_stopsmtp system procedure” on page 1344

Example

This example gets the most recent SMTP or MAPI message text.

```
SELECT xp_get_mail_error_text()
```

This example uses SMTP to initiate the sending of a plain text message.

```
BEGIN
    DECLARE err_smtp INTEGER;
    DECLARE err_code INTEGER;
    DECLARE err_msg  LONG VARCHAR;

    SELECT xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' ) INTO err_smtp;
    SELECT xp_get_mail_error_code( ), xp_get_mail_error_text( ) INTO err_code, err_msg;
    SELECT err_smtp, err_code, err_msg;
END;
```
xp_getenv system procedure

Returns the value of an environment variable.

Syntax

xp_getenv( environment_variable )

Arguments

- **environment_variable**  Use this VARCHAR(8000) parameter to specify the environment variable. This parameter is case insensitive on Windows operating systems and case sensitive on all other operating systems, independent of the case sensitivity of the database. The default value is NULL.

Returns

This function returns a LONG BINARY value.

Remarks

If the environment variable specified is NULL or not set, NULL is returned.

Privileges

You must have the SERVER OPERATOR system privilege.

Side effects

None

See also

- “SQL Anywhere environment variables” [SQL Anywhere Server - Database Administration]

Example

The following example uses the xp_getenv system procedure to return the value of the environment variable PATH.

```
SELECT CAST(xp_getenv('PATH') AS LONG VARCHAR);
```

The following example uses the xp_getenv and sa_split_list system procedures to return the value of the Windows environment variable PATH as a list. Use ':' as the separator character on Unix operating systems.

```
CALL sa_split_list(CAST(xp_getenv('PATH') AS LONG VARCHAR), ';
```

The following example uses the environment variable NONEXISTENT, which is assumed to not exist. Therefore, the query is expected to return NULL.

```
SELECT xp_getenv( 'NONEXISTENT' );
```
**xp_msver system procedure**

Retrieves version and name information about the database server.

**Syntax**

```sql
xp_msver( the_option )
```

**Arguments**

- **the_option**
  
  Use this CHAR(254) parameter to specify a string. The string must be one of the following, enclosed in string delimiters.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductName</td>
<td>Returns the name of the product (SQL Anywhere). ProductName is the default argument value.</td>
</tr>
<tr>
<td>ProductVersion</td>
<td>Returns the version number, followed by the build number. The format is as follows:</td>
</tr>
<tr>
<td></td>
<td>16.0.0.1403</td>
</tr>
<tr>
<td>CompanyName</td>
<td>Returns the name of the company.</td>
</tr>
<tr>
<td>FileDescription</td>
<td>Returns the name of the product, followed by the name of the operating system.</td>
</tr>
<tr>
<td>LegalCopyright</td>
<td>Returns a copyright string for the software.</td>
</tr>
<tr>
<td>LegalTrademarks</td>
<td>Returns trademark information for the software.</td>
</tr>
</tbody>
</table>

**Returns**

This function returns a CHAR(254) value.

**Remarks**

This function returns product, company, version, and other information.

**Privileges**

None

**See also**

- “System functions” on page 154

**Example**

The following statement requests the version and operating system description:

```sql
SELECT xp_msver( 'ProductVersion' ) Version,
       xp_msver( 'FileDescription' ) Description;
```
Sample output is as follows. The value for Version will likely be different on your system.

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.0.0.1403</td>
<td>SQL Anywhere Windows XP</td>
</tr>
</tbody>
</table>

**xp_read_file system procedure**

Reads a file and returns the contents of the file as a LONG BINARY variable.

**Syntax**

```sql
xp_read_file(
    filename
    [, lazy ]
)
```

**Arguments**

- **filename**  Use this LONG VARCHAR parameter to specify the name of the file for which to return the contents.

- **lazy**  When you specify this optional INTEGER parameter and its value is not 0, the contents of the file are not read until they are requested. Reads only occur when the LONG BINARY value is accessed and only on the portion of the file that is requested. The default is 0, or non-lazy.

**Returns**

This function returns the contents of the named file as a LONG BINARY value. If the file does not exist or cannot be read, NULL is returned.

**Remarks**

The `filename` is relative to the starting directory of the database server.

The function can be useful for inserting entire documents or images stored in files into tables. If the file cannot be read, the function returns NULL.

If the data file is in a different character set, you can use the CSCONVERT function to convert it.

You can also use the CSCONVERT function to address character set conversion requirements you have when using the `xp_read_file` system procedure.

**Privileges**

You must have the READ FILE system privilege.

**Example**

The following statement inserts an image into a column named Photo of the Products table.

```sql
UPDATE Products 
SET Photo=xp_read_file( 'c:\sqlany\scripts\adata\HoodedSweatshirt.jpg' ) 
WHERE Products.ID=600;
```
The following statement reads a text file and displays each line with a line number.

```
SELECT * FROM sa_split_list( CAST(xp_read_file(',\\Windows\\win.ini') AS LONG VARCHAR), 0x0a);
```

See also
- “CSCONVERT function [String]” on page 198
- “xp_write_file system procedure” on page 1344
- “Directory and file system procedures” on page 1088
- “CALL statement” on page 530
- “Using OPENXML with xp_read_file” [SQL Anywhere Server - SQL Usage]
- “Insertion of documents and images” [SQL Anywhere Server - SQL Usage]

**xp_scanf system procedure**

Extracts substrings from an input string and a format string.

**Syntax**

```
xp_scanf(
    input_buffer
    , format
)
```

**Arguments**

- **input_buffer** Use this CHAR(254) parameter to specify the input string.
- **format** Use this CHAR(254) parameter to specify the format of the input string, using place holders (%s) for each parm argument. There can be up to fifty place holders in the format argument, and there must be the same number of place holders as parm arguments.

**Result set**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parm</td>
<td>CHAR(254)</td>
<td>Returns substrings extracted from input_buffer using the specified format.</td>
</tr>
</tbody>
</table>

**Privileges**

None

**See also**

- “CALL statement” on page 530

**Example**

The following statements extract the substrings Hello and World! from the input buffer Hello World!, and put them into variables string1 and string2, and then selects them:
xp_sendmail system procedure

Sends an email message.

Syntax

xp_sendmail(
    recipient = mail-address
[, subject = subject ]
[, cc_recipient = mail-address ]
[, bcc_recipient = mail-address ]
[, query = sql-query ]
[, "message" = message-body ]
[, attachment = attach-name ]
[, attach_result = attach-result ]
[, echo_error = echo-error ]
[, include_file = filename ]
[, no_column_header = no-column-header ]
[, no_output = no-output ]
[, width = width ]
[, separator = separator-char ]
[, dbuser = user-name ]
[, dbname = db-name ]
[, type = type ]
[, include_query = include-query ]
[, content_type = content-type ]
)

Arguments

Some arguments supply fixed values and are available for use to ensure Transact-SQL compatibility, as noted below.

- **recipient**  This LONG VARCHAR parameter specifies the recipient mail address. When specifying multiple recipients, each mail address must be separated by a semicolon.

- **subject**  This LONG VARCHAR parameter specifies the subject field of the message. The default is NULL.

- **cc_recipient**  This LONG VARCHAR parameter specifies the cc recipient mail address. When specifying multiple cc recipients, each mail address must be separated by a semicolon. The default is NULL.

- **bcc_recipient**  This LONG VARCHAR parameter specifies the bcc recipient mail address. When specifying multiple bcc recipients, each mail address must be separated by a semicolon. The default is NULL.

- **query**  This LONG VARCHAR is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is NULL.
- "message" This LONG VARCHAR parameter specifies the message contents. The default is NULL. The "message" parameter name requires double quotes around it because MESSAGE is a reserved word.

- **attachname** This LONG VARCHAR parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is NULL.

- **attach_result** This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 0.

- **echo_error** This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 1.

- **include_file** This LONG VARCHAR parameter specifies an attachment file. The default is NULL.

- **no_column_header** This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 0.

- **no_output** This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 0.

- **width** This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 80.

- **separator** This CHAR(1) parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is CHAR(9).

- **dbuser** This LONG VARCHAR parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is guest.

- **dbname** This LONG VARCHAR parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is master.

- **type** This LONG VARCHAR parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is NULL.

- **include_query** This INTEGER parameter is provided for Transact-SQL compatibility. It is not used by SQL Anywhere. The default is 0.

- **content_type** This LONG VARCHAR parameter specifies the content type for the "message" parameter (for example, text/html, ASIS, and so on). The default is NULL. The value of content_type is not validated; setting an invalid content type results in an invalid or incomprehensible email being sent.

To set headers manually, set the content_type parameter to ASIS. When you do this, the xp_sendmail procedure assumes that the data passed to the message parameter is a properly formed email with headers, and does not add any additional headers. When specifying ASIS, you must set all the headers manually in the message parameter, even headers that would normally be filled in by passing data to the other parameters.
Returns

This function returns an INTEGER status code.

See “Return codes for MAPI and SMTP system procedures” on page 1086.

Remarks

xp_sendmail is a system procedure that sends an email message to the specified recipients once a session has been started with xp_startmail or xp_startsmtp. The procedure accepts messages of any length. The argument values for xp_sendmail are strings. The length of each argument is limited to the amount of available memory on your system.

The content_type argument is intended for users who understand the requirements of MIME email. xp_sendmail accepts ASIS as a content_type. When content_type is set to ASIS, xp_sendmail assumes that the message body ("message") is a properly formed email with headers, and does not add any additional headers. Specify ASIS to send multipart messages containing more than one content type. For more information about MIME, see RFCs 2045-2049 (http://www.ietf.org/).

Any attachment specified by the include_file parameter is sent as application/octet-stream MIME type, with base64 encoding, and must be present on the database server.

In SQL Anywhere 10.0.0 and later, email sent with an SMTP email system is encoded if the subject line contains characters that are not 7-bit ASCII. Also, email sent to an SMS-capable device may not be decoded properly if the subject line contains characters that are not 7-bit ASCII.

Must have executed xp_startmail to start an email session using MAPI, or xp_startsmtp to start an email session using SMTP.

If you are sending mail using MAPI, the content_type parameter is not supported.

Privileges

You must have the SEND EMAIL system privilege.

See also

● “MAPI and SMTP system procedures” on page 1086
● “xp_startmail system procedure” on page 1340
● “xp_startsmtp system procedure” on page 1341
● “xp_stopmail system procedure” on page 1343
● “xp_stopssmtp system procedure” on page 1344
● “CALL statement” on page 530
● “Reserved words” on page 1

Example

This example uses SMTP to send a plain text message.

```sql
CALL xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' );
CALL xp_sendmail( recipient='jane.smith@sample.com',
                  subject='This is my subject line',
                  "message"='This text is the body of my email.\n' );
CALL xp_stopssmtp( );
```
This example uses SMTP to send an HTML formatted message with an attachment.

```sql
CALL xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' );
CALL xp_sendmail( recipient='jane.smith@sample.com',
subject='HTML mail example with attachment',
"message"='Plain text.<BR><BR><B>Bold text.</B><BR><BR>' ||
'<a href="www.sap.com">SAP Home Page</a>',
content_type = 'text/html',
include_file = '\temp\sendmail2.sql' );
CALL xp_stopsmtp();
```

This example uses SMTP to send an inline HTML formatted message with an attachment.

```sql
CALL xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' );
CALL xp_sendmail( recipient='jane.smith@sample.com',
subject='Inline HTML mail example with attachment',
"message"="Content-Type: text/html;\nContent-Disposition: inline;\n
' ||
'Plain text.<BR><BR><B>Bold text.</B><BR><BR>' ||
'<a href="www.sap.com">SAP Home Page</a>',
content_type = 'ASIS',
include_file = '\temp\sendmail3.sql' );
CALL xp_stopsmtp();
```

This example uses SMTP to send an inline HTML formatted message with a signature and two attachments, one of which is a ZIP file.

```sql
BEGIN
DECLARE content LONG VARCHAR;
SET content =
'Content-Type: multipart/mixed; boundary="xxxxx";\nThis part of the email should not be shown. If this ';' is shown then the email client is not MIME compatible\n|--xxxxx\n|-- Content-Type: text/html;\nContent-Disposition: inline;\n' ||
'Plain text.<BR><BR><B>Bold text.</B><BR><BR>' ||
'<a href="www.sap.com">SAP Home Page</a>\n' ||
xp_read_file( '\temp\johndoe.sig.html' ) ||
|--xxxxx\n|-- Content-Type: application/zip; name="sendmail4.zip"\n' ||
'Content-Transfer-Encoding: base64\n' ||
'Content-Disposition: attachment; filename="sendmail4.zip"\nbase64_encode( xp_read_file( '\temp\sendmail4.zip' ) ) ||
'\n' ||
|--xxxxx--
';
CALL xp_startsmtp( 'doe@sample.com', 'corporatemail.sample.com' );
CALL xp_sendmail( recipient='jane.smith@sample.com',
subject='Inline HTML mail example with signature and 2 attachments',
"message"=content,
content_type = 'ASIS',
include_file = '\temp\sendmail4.sql' );
CALL xp_stopsmtp();
END
```

**xp_sprintf system procedure**

Builds a result string from a set of input strings.
Syntax

xp_sprintf(
    format,
    parm [, parm2 ... ]
)

Arguments

- **format** Use this CHAR(254) parameter to specify how to format the result string, using place holders (%s) for each `parm` argument. There can be up to fifty place holders in the `format` argument, and there should be the same number of place holders as `parm` arguments.

- **parm** The input strings that are used in the result string. You can specify up to 50 of these CHAR(254) arguments.

Result set

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output_buffer</td>
<td>CHAR(254)</td>
<td>Returns a result string built from the <code>format</code> and <code>parm</code> arguments.</td>
</tr>
</tbody>
</table>

Privileges

None

See also

- “CALL statement” on page 530

Example

The following statements put the string Hello World! into the result variable.

```sql
CREATE VARIABLE result CHAR( 254);
CALL xp_sprintf( result, '%s %s', 'Hello', 'World!' );
```

**xp_startmail system procedure**

Starts an email session under MAPI.

Syntax

xp_startmail(
    [ mail_user = mail-login-name ]
    [, mail_password = mail-password ]
)

Arguments

- **mail_user** Use this LONG VARCHAR parameter to specify the MAPI login name. The default is NULL.

- **mail_password** Use this LONG VARCHAR parameter to specify the MAPI password. The default is NULL.
Returns
This function returns an INTEGER status code.

See “Return codes for MAPI and SMTP system procedures” on page 1086.

Remarks
xp_startmail is a system procedure that starts an email session.

If you are using Microsoft Exchange, the mail-login-name argument is an Exchange profile name, and you should not include a password in the procedure call.

Not supported on Unix.

Privileges
You must have the SEND EMAIL system privilege.

See also
● “MAPI and SMTP system procedures” on page 1086
● “xp_stopmail system procedure” on page 1343
● “xp_sendmail system procedure” on page 1336
● “xp_startsmtp system procedure” on page 1341
● “xp_stopsmtp system procedure” on page 1344
● “CALL statement” on page 530
● “General security tips” [SQL Anywhere Server - Database Administration]

xp_startsmtp system procedure
Starts an email session under SMTP.

Syntax
xp_startsmtp(  
smtp_sender = email-address  
, smtp_server = smtp-server  
[, smtp_port = port-number ]  
[, timeout = timeout ]  
[, smtp_sender_name = username ]  
[, smtp_auth_username = auth-username ]  
[, smtp_auth_password = auth-password ]  
[, trusted_certificates = public-certificate ]  
[, certificate_company = organization ]  
[, certificate_unit = organization-unit ]  
[, certificate_name = common-name ] )

Arguments
● smtp_sender   This LONG VARCHAR parameter specifies the email address of the sender.
● smtp_server   This LONG VARCHAR parameter specifies which SMTP server to use, and is the server name or IP address.
smtp_port  This optional INTEGER parameter specifies the port number to connect to on the SMTP server. The default is 25.

timeout  This optional INTEGER parameter specifies how long to wait, in seconds, for a response from the database server before aborting the current call to `xp_sendmail`. The default is 60 seconds.

smtp_sender_name  This optional LONG VARCHAR parameter specifies an alias for the sender's email address. For example, 'JSmith' instead of 'email-address'. The default is NULL.

smtp_auth_username  This optional LONG VARCHAR parameter specifies the user name to provide to SMTP servers requiring authentication. The default is NULL.

smtp_auth_password  This optional LONG VARCHAR parameter specifies the password to provide to SMTP servers requiring authentication. The default is NULL.

trusted_certificates  This optional LONG VARCHAR parameter specifies a key/value pair or the path and file name of a file that contains one or more trusted certificates. The default is NULL. When this parameter is NULL, a standard SMTP connection is made.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>file=</td>
<td>The path and file name of a file that contains one or more trusted certificates.</td>
</tr>
<tr>
<td>cert_name=</td>
<td>The name of a certificate stored in the database.</td>
</tr>
</tbody>
</table>

The trusted certificate can be a server's self-signed certificate, a public enterprise root certificate, or a certificate belonging to a commercial Certificate Authority. You must generate your certificates using RSA.

certificate_company  This optional LONG VARCHAR parameter specifies that the client accepts server certificates only when the Organization field of the certificate matches this value. This parameter is ignored when the trusted_certificates value is NULL. The default is NULL.

certificate_unit  This optional LONG VARCHAR parameter specifies that the client accepts server certificates only when the Organization Unit field of the certificate matches this value. This parameter is ignored when the trusted_certificates value is NULL. The default is NULL.

certificate_name  This optional LONG VARCHAR parameter specifies that the client accepts server certificates only when the Common Name field on the certificate matches this value. This parameter is ignored when the trusted_certificates value is NULL. The default is NULL.

Returns

This function returns an INTEGER status code.

See “Return codes for MAPI and SMTP system procedures” on page 1086.
Remarks

xp_startsmtp is a system procedure that starts a mail session for a specified email address by connecting to an SMTP server. This connection can time out. Therefore, it is recommended that you call xp_startsmtp just before executing xp_sendmail.

If you specify smtp_auth_username and smtp_auth_password, and the server does not support the SMTP authentication capability, error code 104 is returned.

Virus scanners can affect xp_startsmtp, causing it to return error code 100. For McAfee VirusScan version 8.0.0 and later, settings for preventing mass mailing of email worms also prevent xp_sendmail from executing properly. If your virus scanning software allows you to specify processes that can bypass the mass mailing protections, specify `dbeng16.exe` and `dbsrv16.exe`. For example, with McAfee VirusScan you can prevent mass mailing by adding these two processes to the list of Excluded Processes in the Properties area.

Privileges

You must have the SEND EMAIL system privilege.

See also

- “MAPI and SMTP system procedures” on page 1086
- “xp_startmail system procedure” on page 1340
- “xp_stopmail system procedure” on page 1343
- “xp_sendmail system procedure” on page 1336
- “xp_stopsmtmp system procedure” on page 1344
- “CALL statement” on page 530
- “CREATE CERTIFICATE statement” on page 547
- “General security tips” [SQL Anywhere Server - Database Administration]

xp_stopmail system procedure

Closes a MAPI email session.

Syntax

```
xp_stopmail()
```

Returns

This function returns an INTEGER status code.

See “Return codes for MAPI and SMTP system procedures” on page 1086.

Remarks

xp_stopmail is a system procedure that ends an email session.

Not supported on Unix.
Privileges

You must have the SEND EMAIL system privilege.

See also

- “MAPI and SMTP system procedures” on page 1086
- “xp_startmail system procedure” on page 1340
- “xp_sendmail system procedure” on page 1336
- “xp_startsmtp system procedure” on page 1341
- “xp_stopsmtp system procedure” on page 1344
- “CALL statement” on page 530
- “General security tips” [SQL Anywhere Server - Database Administration]

xp_stopsmtp system procedure

Closes an SMTP email session.

Syntax

xp_stopsmtp( )

Returns

This function returns an INTEGER status code.

See “Return codes for MAPI and SMTP system procedures” on page 1086.

Remarks

xp_stopsmtp is a system procedure that ends an email session.

Privileges

You must have the SEND EMAIL system privilege.

See also

- “MAPI and SMTP system procedures” on page 1086
- “xp_startmail system procedure” on page 1340
- “xp_stopmail system procedure” on page 1343
- “xp_sendmail system procedure” on page 1336
- “xp_startsmtp system procedure” on page 1341
- “CALL statement” on page 530
- “General security tips” [SQL Anywhere Server - Database Administration]

xp_write_file system procedure

Writes data to a file from a SQL statement.
**Syntax**

```
xp_write_file(
    filename,  
    file_contents
)
```

**Arguments**

- **filename**  Use this LONG VARCHAR parameter to specify the file name.
- **file_contents**  Use this LONG BINARY parameter to specify the contents to write to the file.

**Returns**

This function returns an INTEGER status code.

**Remarks**

The function writes `file_contents` to the file `filename`. It returns 0 if successful, and non-zero if it fails.

The `filename` value can be prefixed by either an absolute or a relative path. If `filename` is prefixed by a relative path, then the file name is relative to the current working directory of the database server. If the file already exists, its contents are overwritten.

This function can be useful for unloading long binary data into files.

You can also use the CSCONVERT function to address character set conversion requirements you have when using the `xp_write_file` system procedure.

**Privileges**

You must have the WRITE FILE system privilege.

**See also**

- “CSCONVERT function [String]” on page 198
- “xp_read_file system procedure” on page 1334
- “Insertion of documents and images”  [SQL Anywhere Server - SQL Usage]
- “Directory and file system procedures” on page 1088

**Examples**

This example uses `xp_write_file` to create a file `accountnum.txt` containing the data `123456`:

```
CALL xp_write_file( 'accountnum.txt', '123456' );
```

This example queries the Contacts table of the sample database, and then creates a text file for each contact living in New Jersey. Each text file is named using a concatenation of the contact’s first name (GivenName), last name (Surname), and then the string `.txt` (for example, `Reeves_Scott.txt`), and contains the contact’s street address (Street), city (City), and state (State), on separate lines.

```
SELECT xp_write_file(
    Surname || '_' ||  GivenName || '.txt',
    Street || '
' || City || '
' || State )
FROM Contacts WHERE State = 'NJ';
```
This example uses `xp_write_file` to create an image file (JPG) for every product in the Products table. Each value of the ID column becomes a file name for a file with the contents of the corresponding value of the Photo column:

```
SELECT xp_write_file( ID || '.jpg', Photo ) FROM Products;
```

In the example above, ID is a row with a UNIQUE constraint. This is important to ensure that a file isn't overwritten with the contents of subsequent row. Also, you must specify the file extension applicable to the data stored in the column. In this case, the Products.Photo stores image data (JPEGs).
Views

System views

The catalog contains system tables that link together by keys and indexes. In SQL Anywhere, the system tables are hidden. However, there is a system view for each table. A system view can also include columns from more than one system table to satisfy a commonly needed join.

To ensure compatibility with future versions of the SQL Anywhere catalog, applications must make use of system views and not the underlying system tables, which may change.

Viewing detailed system information for views and definitions

Database administrators can access information about system views, including their definitions, from Sybase Central.

Prerequisites

There are no prerequisites for this task.

Context and remarks

System views are updated when a checkpoint occurs.

Task

1. Use the SQL Anywhere 16 plug-in to connect to the database.
2. Right-click the database, and then click Configure Owner Filter.
3. Click SYS, and then click OK.
4. In the left pane, double-click Views.
5. In the left pane click a view owned by SYS (this is indicated by the phrase SYS in parenthesis after the name of the view), and in the right pane click the SQL tab.
6. Click the Data tab.

Results

The view definition appears on the Data tab.
SYSARTICLE system view

Each row of the SYSARTICLE system view describes an article in a publication. The underlying system table for this view is ISYSARTICLE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>UNSIGNED INT</td>
<td>The publication of which the article is a part.</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>Each article consists of columns and rows from a single table. This column contains the table ID for this table.</td>
</tr>
<tr>
<td>where_expr</td>
<td>LONG VARCHAR</td>
<td>For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.</td>
</tr>
<tr>
<td>subscribe_by_expr</td>
<td>LONG VARCHAR</td>
<td>For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.</td>
</tr>
<tr>
<td>query</td>
<td>CHAR(1)</td>
<td>Indicates information about the article type to the database server.</td>
</tr>
<tr>
<td>alias</td>
<td>VARCHAR(256)</td>
<td>The alias for the article.</td>
</tr>
<tr>
<td>schema_change_active</td>
<td>BIT</td>
<td>1 if the table and publication are part of a synchronization schema change.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (publication_id, table_id)
- FOREIGN KEY (publication_id) REFERENCES SYS.ISYSPUBLICATION (publication_id)
- FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)

SYSARTICLECOL system view

Each row of the SYSARTICLECOL system view identifies a column in an article. The underlying system table for this view is ISYSARTICLECOL.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifier for the publication of which the column is a part.</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>The table to which the column belongs.</td>
</tr>
<tr>
<td>column_id</td>
<td>UNSIGNED INT</td>
<td>The column identifier, from the SYSTABCOL system view.</td>
</tr>
</tbody>
</table>
Constraints on underlying system table

+ PRIMARY KEY (publication_id, table_id, column_id)

+ FOREIGN KEY (publication_id, table_id) REFERENCES SYS.ISYSARTICLE (publication_id, table_id)

+ FOREIGN KEY (table_id, column_id) REFERENCES SYS.ISYSTABCOL (table_id, column_id)

**SYSCAPABILITY system view**

Each row of the SYSCAPABILITY system view specifies the status of a capability on a remote database server. The underlying system table for this view is ISYSCAPABILITY.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capid</td>
<td>INTEGER</td>
<td>The ID of the capability, as listed in the SYSCAPABILITY-NAME system view.</td>
</tr>
<tr>
<td>srvid</td>
<td>UNSIGNED INT</td>
<td>The server to which the capability applies, as listed in the SYS-SERVER system view.</td>
</tr>
<tr>
<td>capvalue</td>
<td>CHAR(128)</td>
<td>The value of the capability.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

+ PRIMARY KEY (capid, srvid)

+ FOREIGN KEY (srvid) REFERENCES SYS.ISYSSERVER (srvid)

See also

- “SYSCAPABILITYNAME system view” on page 1349

**SYSCAPABILITYNAME system view**

Each row in the SYSCAPABILITYNAME system view provides a name for each capability ID in the SYSCAPABILITY system view.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capid</td>
<td>INTEGER</td>
<td>A number uniquely identifying the capability.</td>
</tr>
<tr>
<td>capname</td>
<td>VARCHAR(32000)</td>
<td>The name of the capability.</td>
</tr>
</tbody>
</table>
Remarks
The SYSCAPABILITYNAME system view is defined using a combination of sa_rowgenerator and the following server properties:

- RemoteCapability
- MaxRemoteCapability

See also
- “List of database server properties” [SQL Anywhere Server - Database Administration]
- “SYSCAPABILITY system view” on page 1349

SYSCERTIFICATE system view
Each row of the SYSCERTIFICATE system view stores a certificate in text PEM-format. The underlying system table for this view is ISYSCERTIFICATE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the certificate.</td>
</tr>
<tr>
<td>cert_name</td>
<td>CHAR(128)</td>
<td>The certificate name.</td>
</tr>
<tr>
<td>contents</td>
<td>LONG BINARY</td>
<td>The certificate contents in a compressed form.</td>
</tr>
<tr>
<td>update_time</td>
<td>TIMESTAMP</td>
<td>The local date and time of the last create or replace.</td>
</tr>
<tr>
<td>update_time_utc</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>The UTC date and time of the last create or replace.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table
- PRIMARY KEY (object_id)
- UNIQUE INDEX (cert_name)

SYSCHECK system view
Each row in the SYSCHECK system view provides the definition for a named check constraint in a table. The underlying system table for this view is ISYSCHECK.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_id</td>
<td>UNSIGNED INT</td>
<td>A number that uniquely identifies the constraint in the database.</td>
</tr>
<tr>
<td>check_defn</td>
<td>LONG VARCHAR</td>
<td>The CHECK expression.</td>
</tr>
</tbody>
</table>
Constraints on underlying system table

**PRIMARY KEY** (check_id)

**FOREIGN KEY** (check_id) REFERENCES SYS.ISYSCONSTRAINT (constraint_id)

**SYSCOLPERM system view**

The GRANT statement can give UPDATE, SELECT, or REFERENCES privileges to individual columns in a table. Each column with UPDATE, SELECT, or REFERENCES privileges is recorded in one row of the SYSCOLPERM system view. The underlying system table for this view is ISYSCOLPERM.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>The table number for the table containing the column.</td>
</tr>
<tr>
<td>grantee</td>
<td>UNSIGNED INT</td>
<td>The user number of the user ID that is given privilege on the column. If the grantee the PUBLIC role, the privilege is given to all user IDs.</td>
</tr>
<tr>
<td>grantor</td>
<td>UNSIGNED INT</td>
<td>The user number of the user ID that grants the privilege.</td>
</tr>
<tr>
<td>column_id</td>
<td>UNSIGNED INT</td>
<td>This column number, together with the table_id, identifies the column for which privilege has been granted.</td>
</tr>
<tr>
<td>privilege_type</td>
<td>SMALLINT</td>
<td>The number in this column indicates the kind of column privilege (16=REFERENCES, 1=SELECT, or 8=UPDATE).</td>
</tr>
<tr>
<td>is_grantable</td>
<td>CHAR(1)</td>
<td>Indicates if the privilege on the column was granted WITH GRANT OPTION.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

**PRIMARY KEY** (table_id, grantee, grantor, column_id, privilege_type)

**FOREIGN KEY** (table_id, column_id) REFERENCES SYS.ISYSTABCOL (table_id, column_id)

**FOREIGN KEY** (grantor) REFERENCES SYS.ISYSUSER (user_id)

**FOREIGN KEY** (grantee) REFERENCES SYS.ISYSUSER (user_id)

**SYSCOLSTAT system view**

The SYSCOLSTAT system view contains the column statistics, including histograms, that are used by the optimizer. The contents of this view are best retrieved using the sa_get_histogram stored procedure or the Histogram utility. The underlying system table for this view is ISYSCOLSTAT.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>A number that uniquely identifies the table or materialized view to which the column belongs.</td>
</tr>
<tr>
<td>column_id</td>
<td>UNSIGNED INT</td>
<td>A number that, together with table_id, uniquely identifies the column.</td>
</tr>
<tr>
<td>format_id</td>
<td>SMALLINT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>update_time</td>
<td>TIMESTAMP</td>
<td>The local time of the last update of the column statistics.</td>
</tr>
<tr>
<td>density</td>
<td>FLOAT</td>
<td>An estimate of the average selectivity of a single value for the column, not counting the large single value selectivities stored in the row.</td>
</tr>
<tr>
<td>max_steps</td>
<td>SMALLINT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>actual_steps</td>
<td>SMALLINT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>step_values</td>
<td>LONG BINARY</td>
<td>For system use only.</td>
</tr>
<tr>
<td>frequencies</td>
<td>LONG BINARY</td>
<td>For system use only.</td>
</tr>
<tr>
<td>update_time_utc</td>
<td>TIMESTAMP</td>
<td>The UTC time of the last update of the column statistics.</td>
</tr>
</tbody>
</table>

Note
For databases created using SQL Anywhere 16 or later, the underlying system table for this view is always encrypted to protect the data from unauthorized access.

Constraints on underlying system table

<table>
<thead>
<tr>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIMARY KEY (table_id, column_id)</td>
</tr>
<tr>
<td>FOREIGN KEY (table_id, column_id) REFERENCES SYS.ISYSTABCOL (table_id, column_id)</td>
</tr>
</tbody>
</table>

See also
- “sa_get_histogram system procedure” on page 1145
- “Histogram utility (dbhist)” [SQL Anywhere Server - Database Administration]

SYSCONSTRAINT system view

Each row in the SYSCONSTRAINT system view describes a named constraint in the database. The underlying system table for this view is ISYSCONSTRAINT.
Constraints on underlying system table

```sql
PRIMARY KEY (constraint_id)
FOREIGN KEY (ref_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
FOREIGN KEY (table_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
UNIQUE CONSTRAINT (table_object_id, constraint_name)
```

**SYSDBFILE system view**

Each row in the SYSDBFILE system view describes a dbspace file. The underlying system table for this view is ISYSDBFILE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbfile_id</td>
<td>SMALLINT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>dbspace_id</td>
<td>SMALLINT</td>
<td>Each dbspace file in a database is assigned a unique number. The system dbspace contains all system objects and has a dbspace_id of 0.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>dbfile_name</td>
<td>CHAR(128)</td>
<td>The file name for the dbspace.</td>
</tr>
<tr>
<td>file_name</td>
<td>LONG VARCHAR</td>
<td>A unique name for the dbspace. It is used in the CREATE TABLE command.</td>
</tr>
<tr>
<td>lob_map</td>
<td>LONG VARBIT</td>
<td>For internal use only.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (dbfile_id)
- FOREIGN KEY (dbspace_id) REFERENCES SYS.ISYSDBSPACE (dbspace_id)
- UNIQUE index (file_name)

**SYSDBSPACE system view**

Each row in the SYSDBSPACE system view describes a dbspace file. The underlying system table for this view is ISYSDBSPACE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbspace_id</td>
<td>SMALLINT</td>
<td>Unique number identifying the dbspace. The system dbspace contains all system objects and has a dbspace_id of 0.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the dbspace.</td>
</tr>
<tr>
<td>dbspace_name</td>
<td>CHAR(128)</td>
<td>A unique name for the dbspace. It is used in the CREATE TABLE command.</td>
</tr>
<tr>
<td>store_type</td>
<td>TINYINT</td>
<td>For internal use only.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (dbspace_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

**SYSDBSPACEPERM system view**

Each row in the SYSDBSPACEPERM system view describes a privilege on a dbspace file. The underlying system table for this view is ISYSDBSPACEPERM.
Column name | Data type | Description
---|---|---
dbspace_id | SMALLINT | Unique number identifying the dbspace. The system dbspace contains all system objects and has a dbspace_id of 0.
grantee | UNSIGNED INT | The user ID of the user getting the privilege.
privilege_type | SMALLINT | The privilege that is granted to the grantee. For example, CREATE gives the grantee privilege to create objects on the dbspace.

**Constraints on underlying system table**

FOREIGN KEY (dbspace_id) REFERENCES SYS.ISYSDBSPACE (dbspace_id)

FOREIGN KEY (grantee) REFERENCES SYS.ISYSUSER (user_id)

**See also**

- “GRANT statement” on page 827
- “User security (roles and privileges)” [SQL Anywhere Server - Database Administration]

---

**SYSDEPENDENCY system view**

Each row in the SYSDEPENDENCY system view describes a dependency between two database objects. The underlying system table for this view is ISYSDEPENDENCY.

A dependency exists between two database objects when one object references another object in its definition. For example, if the query specification for a view references a table, the view is dependent on the table. The database server tracks dependencies of views on tables, views, materialized views, and columns.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the referenced object.</td>
</tr>
<tr>
<td>dep_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the referencing object.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

PRIMARY KEY (ref_object_id, dep_object_id)

FOREIGN KEY (ref_object_id) REFERENCES SYS.ISYSOBJECT (object_id)

FOREIGN KEY (dep_object_id) REFERENCES SYS.ISYSOBJECT (object_id)

**See also**

- “sa_dependent_views system procedure” on page 1123
- “View dependencies” [SQL Anywhere Server - SQL Usage]
**SYSDOMAIN system view**

The SYSDOMAIN system view records information about built-in data types (also called domains). The contents of this view does not change during normal operation. The underlying system table for this view is ISYSDOMAIN.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain_id</td>
<td>SMALLINT</td>
<td>The unique number assigned to each data type. These numbers cannot be changed.</td>
</tr>
<tr>
<td>domain_name</td>
<td>CHAR(128)</td>
<td>The name of the data type normally found in the CREATE TABLE command, such as CHAR or INTEGER.</td>
</tr>
<tr>
<td>type_id</td>
<td>SMALLINT</td>
<td>The ODBC data type. This value corresponds to the value for data_type in the Transact-SQL compatibility dbo.SYSTYPES table.</td>
</tr>
<tr>
<td>&quot;precision&quot;</td>
<td>SMALLINT</td>
<td>The number of significant digits that can be stored using this data type. The column value is NULL for non-numeric data types.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

 PRIMARY KEY (domain_id)

**SYSEVENT system view**

Each row in the SYSEVENT system view describes an event created with CREATE EVENT. The underlying system table for this view is ISYSEVENT.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_id</td>
<td>UNSIGNED INT</td>
<td>The unique number assigned to each event.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The internal ID for the event, uniquely identifying it in the database.</td>
</tr>
<tr>
<td>creator</td>
<td>UNSIGNED INT</td>
<td>The user number of the owner of the event. The name of the user can be found by looking in the SYSUSER system view.</td>
</tr>
<tr>
<td>event_name</td>
<td>VARCHAR(128)</td>
<td>The name of the event.</td>
</tr>
<tr>
<td>enabled</td>
<td>CHAR(1)</td>
<td>Indicates whether the event is allowed to fire.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>location</td>
<td>CHAR(1)</td>
<td>The location where the event is to fire:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Y = AT ALL clause and FOR PRIMARY clause specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● E = AT CONSOLIDATED clause and FOR PRIMARY clause specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● T = AT REMOTE clause and FOR PRIMARY clause specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● P = (AT clause not specified) FOR PRIMARY clause specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● B = AT ALL clause and FOR ALL clause specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● D = AT CONSOLIDATED clause and FOR ALL clause specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● S = AT REMOTE clause and FOR ALL clause specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● M = (AT clause not specified) FOR ALL clause specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● C = AT CONSOLIDATED (FOR clause not specified)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● R = AT REMOTE (FOR clause not specified)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● A = AT ALL clause (FOR clause not specified)</td>
</tr>
<tr>
<td>event_type_id</td>
<td>UNSIGNED INT</td>
<td>For system events, the event type as listed in the SYSEVENT-TYPE system view.</td>
</tr>
<tr>
<td>action</td>
<td>LONG VARCHAR</td>
<td>The event handler definition. An obfuscated value indicates a hidden event.</td>
</tr>
<tr>
<td>external_action</td>
<td>LONG VARCHAR</td>
<td>For system use only.</td>
</tr>
<tr>
<td>condition</td>
<td>LONG VARCHAR</td>
<td>The condition used to control firing of the event handler.</td>
</tr>
<tr>
<td>remarks</td>
<td>LONG VARCHAR</td>
<td>Remarks for the event; this column comes from ISYSREMARK.</td>
</tr>
<tr>
<td>source</td>
<td>LONG VARCHAR</td>
<td>The original source for the event; this column comes from ISYSSSOURCE.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

```
PRIMARY KEY (event_id)
FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
UNIQUE INDEX (event_name)
```
SYSEVENTTYPE system view

The SYSEVENTTYPE system view defines the system event types that can be referenced by CREATE EVENT.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_type_id</td>
<td>INT</td>
<td>The unique number assigned to each event type.</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(32000)</td>
<td>The name of the system event type.</td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR(32000)</td>
<td>A description of the system event type.</td>
</tr>
</tbody>
</table>

Remarks

The SYSEVENTTYPE system view is defined using a combination of sa_rowgenerator and the following server properties:

- EventTypeName
- EventTypeDesc
- MaxEventType

See also

- “List of database server properties” [SQL Anywhere Server - Database Administration]
- “SYSEVENT system view” on page 1356

SYSEXTERNENV system view

Many external runtime environments are supported, including embedded SQL and ODBC applications written in C/C++, and applications written in Java, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

Each row in the SYSEXTERNENV system view describes the information needed to identify and launch each of the external environments. The underlying system table for this view is ISYSEXTERNENV.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique identifier for the external environment.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>name</td>
<td>CHAR(128)</td>
<td>This column identifies the name of the external environment or language. It is one of java, perl, php, clr, c_esql32, c_esql64, c_odbc32, or c_odbc64.</td>
</tr>
<tr>
<td>scope</td>
<td>CHAR(1)</td>
<td>This column is either C for CONNECTION or D for DATABASE respectively. The scope column identifies if the external environment is launched as one-per-connection or one-per-database. For one-per-connection external environments (like PERL, PHP, C_ESQL32, C_ESQL64, C_ODBC32, and C_ODBC64), there is one instance of the external environment for each connection using the external environment. For a one-per-connection, the external environment terminates when the connection terminates. For one-per-database external environments (like JAVA and CLR), there is one instance of the external environment for each database using the external environment. The one-per-database external environment terminates when the database is stopped.</td>
</tr>
<tr>
<td>support_result_sets</td>
<td>CHAR(1)</td>
<td>This column identifies those external environments that can return result sets. All external environments can return result sets except PERL and PHP.</td>
</tr>
<tr>
<td>location</td>
<td>LONG VARCHAR</td>
<td>This column identifies the location on the database server computer where the executable/binary for the external environment can be found. It includes the executable/binary name. This path can either be fully qualified or relative. If the path is relative, then the executable/binary must be in a location where the database server can find it.</td>
</tr>
<tr>
<td>options</td>
<td>LONG VARCHAR</td>
<td>This column identifies any options required on the command line to launch the executable associated with the external environment. You should not modify this column.</td>
</tr>
<tr>
<td>user_id</td>
<td>UNSIGNED INT</td>
<td>When the external environment is initially launched, it must make a connection back to the database to set things up for the external environment's usage. By default, this connection is made using the DBA user ID, but if the database administrator prefers to have the external environment use a different user ID with MANAGE ANY EXTERNAL OBJECT system privilege, then the user_id column would indicate that different user ID instead. Typically, this column is NULL and the database server, by default, uses the DBA user ID.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

```
PRIMARY KEY (object_id)
```
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)

UNIQUE INDEX (name)

See also
- “SQL Anywhere external environment support” [SQL Anywhere Server - Programming]

**SYSEXTERNENVOBJECT system view**

Many external runtime environments are supported, including embedded SQL and ODBC applications written in C/C++, and applications written in Java, Perl, PHP, or languages such as C# and Visual Basic that are based on the Microsoft .NET Framework Common Language Runtime (CLR).

Each row in the SYSEXTERNENVOBJECT system view describes an installed external object. The underlying system table for this view is ISYSEXTERNENVOBJECT.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique identifier for the external object.</td>
</tr>
<tr>
<td>extenv_id</td>
<td>UNSIGNED BIGINT</td>
<td>The unique identifier for the external environment (SYSEXTERNENV.object_id).</td>
</tr>
<tr>
<td>owner</td>
<td>UNSIGNED INT</td>
<td>This column identifies the creator/owner of the external object.</td>
</tr>
<tr>
<td>name</td>
<td>LONG VARCHAR</td>
<td>This column identifies the name of the external object as specified in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSTALL EXTERNAL OBJECT statement.</td>
</tr>
<tr>
<td>contents</td>
<td>LONG BINARY</td>
<td>The contents of the external object.</td>
</tr>
<tr>
<td>update_time</td>
<td>TIMESTAMP</td>
<td>This column identifies the last local time the object was modified (or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>installed).</td>
</tr>
<tr>
<td>update_time_utc</td>
<td>TIMESTAMP WITH TIME</td>
<td>This column identifies the last UTC time the object was modified (or</td>
</tr>
<tr>
<td></td>
<td>ZONE</td>
<td>installed).</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

PRINCIPAL KEY (object_id)

FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (extenv_id) REFERENCES SYS.ISYSEXTERNENV (object_id)

FOREIGN KEY (owner) REFERENCES SYS.ISYSUSER (user_id)
UNIQUE INDEX (name)

See also

- “SQL Anywhere external environment support” [SQL Anywhere Server - Programming]

SYSEXTERNLOGIN system view

Each row in the SYSEXTERNLOGIN system view describes an external login for remote data access. The underlying system table for this view is ISYSEXTERNLOGIN.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>UNSIGNED INT</td>
<td>The user ID on the local database.</td>
</tr>
<tr>
<td>srvid</td>
<td>UNSIGNED INT</td>
<td>The remote server, as listed in the SYSSERVER system view.</td>
</tr>
<tr>
<td>remote_login</td>
<td>VARCHAR(128)</td>
<td>The login name for the user, for the remote server.</td>
</tr>
<tr>
<td>remote_password</td>
<td>VARBINARY(128)</td>
<td>The password for the user, for the remote server.</td>
</tr>
</tbody>
</table>

Note

Previous versions of the catalog contained a SYSEXTERNLOGINS system table. That table has been renamed to be ISYSEXTERNLOGIN (without an 'S'), and is the underlying table for this view.

Note

For databases created using SQL Anywhere 16 or later, the underlying system table for this view is always encrypted to protect the data from unauthorized access.

Constraints on underlying system table

- PRIMARY KEY (user_id, srvid)
- FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)
- FOREIGN KEY (srvid) REFERENCES SYS.ISYSSERVER (srvid)

SYSFKEY system view

Each row in the SYSFKEY system view describes a foreign key constraint in the system. The underlying system table for this view is ISYSFKEY.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>foreign_table_id</td>
<td>UNSIGNED INT</td>
<td>The table number of the foreign table.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>foreign_index_id</td>
<td>UNSIGNED INT</td>
<td>The index number for the foreign key.</td>
</tr>
<tr>
<td>primary_table_id</td>
<td>UNSIGNED INT</td>
<td>The table number of the primary table.</td>
</tr>
<tr>
<td>primary_index_id</td>
<td>UNSIGNED INT</td>
<td>The index number of the primary key.</td>
</tr>
<tr>
<td>match_type</td>
<td>TINYINT</td>
<td>The matching type for the constraint. Matching types include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0  Use the default matching</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1  SIMPLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2  FULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 129 SIMPLE UNIQUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 130 FULL UNIQUE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information about match types, see the MATCH clause of the “CREATE TABLE statement” on page 690.</td>
</tr>
<tr>
<td>check_on_commit</td>
<td>CHAR(1)</td>
<td>Indicates whether INSERT and UPDATE statements should wait until the COMMIT to check if foreign keys are still valid.</td>
</tr>
<tr>
<td>nulls</td>
<td>CHAR(1)</td>
<td>Indicates whether the columns in the foreign key are allowed to contain the NULL value. This setting is independent of the nulls setting in the columns contained in the foreign key.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

- PRIMARY KEY (foreign_table_id, foreign_index_id)
- FOREIGN KEY (foreign_table_id, foreign_index_id) REFERENCES SYS.ISYSIDX (table_id, index_id)
- FOREIGN KEY (primary_table_id, primary_index_id) REFERENCES SYS.ISYSIDX (table_id, index_id)

**SYSHISTORY system view**

Each row in the SYSHISTORY system view records a system operation on the database, such as a database start, a database calibration, and so on. The underlying system table for this view is ISYSHISTORY.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>CHAR(128)</td>
<td>The type of operation performed on the database file. The operation must be one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>INIT</strong> Information about when the database was created.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>UPGRADE</strong> Information about when the database was upgraded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>START</strong> Information about when the database was started using a specific version of the database server on a particular operating system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>LAST_START</strong> Information about the most recent time the database server was started. A LAST_START operation is converted to a START operation when the database is started with a different version of the database server and/or on a different operating system than those values currently stored in the LAST_START row.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>DTT</strong> Information about the second to last Disk Transfer Time (DTT) calibration operation performed on the dbspace. That is, information about the second to last execution of either an ALTER DATABASE CALIBRATE or ALTER DATABASE RESTORE DEFAULT CALIBRATION statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>LAST_DTT</strong> Information about the most recent DTT calibration operation performed on the dbspace. That is, information about the most recent execution of either an ALTER DATABASE CALIBRATE or ALTER DATABASE RESTORE DEFAULT CALIBRATION statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>LAST_BACKUP</strong> Information about the last backup, including date and time of the backup, the backup type, the files that were backed up, and the version of database server that performed the backup.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED INT</td>
<td>For any operation other than DTT and LAST_DTT, the value in this column will be 0. For DTT and LAST_DTT operations, this is the dbspace_id of the dbspace as defined in the SYSDBSPACE system view.</td>
</tr>
</tbody>
</table>
### Column names, Data types, and Description

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
</table>
| sub_operation | CHAR(128)     | For any operation other than DTT and LAST_DTT, the value in this column will be a set of empty single quotes ("."). For DTT and LAST_DTT operations, this column contains the type of sub-operation performed on the dbspace. Values include:  
  - **DTT_SET**  The dbspace calibration has been set.  
  - **DTT_UNSET**  The dbspace calibration has been restored to the default setting.  |
| version      | CHAR(128)     | The version and build number of the database server used to perform the operation.                                                           |
| platform     | CHAR(128)     | The operating system on which the operation was carried out.                                                                                |
| first_time   | TIMESTAMP     | The local date and time the database was first started on a particular operating system with a particular version of the software.           |
| last_time    | TIMESTAMP     | The most recent local date and time the database was started on a particular operating system with a particular version of the software.         |
| details      | LONG VARCHAR  | This column stores information such as command line options used to start the database server or the capability bits enabled for the database.  This information is for use by Technical Support. |
| first_time_utc| TIMESTAMP WITH TIME ZONE | The UTC date and time the database was first started on a particular operating system with a particular version of the software. |
| last_time_utc | TIMESTAMP WITH TIME ZONE | The most recent UTC date and time the database was started on a particular operating system with a particular version of the software. |

**Constraints on underlying system table**

`PRIMARY KEY (operation, object_id, version, platform)`

**See also**

- “SYSDBSPACE system view” on page 1354

### SYSIDX system view

Each row in the SYSIDX system view defines a logical index in the database. The underlying system table for this view is ISYSIDX.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>Uniquely identifies the table to which this index applies.</td>
</tr>
<tr>
<td>index_id</td>
<td>UNSIGNED INT</td>
<td>A unique number identifying the index within its table.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The internal ID for the index, uniquely identifying it in the database.</td>
</tr>
<tr>
<td>phys_index_id</td>
<td>UNSIGNED INT</td>
<td>Identifies the underlying physical index used to implement the logical index. This value is NULL for indexes on temporary tables or remote tables. Otherwise, the value corresponds to the object_id of a physical index in the SYSPHYSIDX system view.</td>
</tr>
<tr>
<td>dbspace_id</td>
<td>SMALLINT</td>
<td>The ID of the file in which the index is contained. This value corresponds to an entry in the SYSDBSPACE system view.</td>
</tr>
<tr>
<td>index_category</td>
<td>TINYINT</td>
<td>The type of index. Values include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1  Primary key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 2  Foreign key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 3  Secondary index (includes unique constraints)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 4  Text indexes</td>
</tr>
<tr>
<td>&quot;unique&quot;</td>
<td>TINYINT</td>
<td>Indicates whether the index is a unique index (1), a unique constraint (2), reserved (3), a non-unique index (4), or a unique index WITH NULLS NOT DISTINCT. A unique index prevents two rows in the indexed table from having the same values in the index columns.</td>
</tr>
<tr>
<td>index_name</td>
<td>CHAR(128)</td>
<td>The name of the index.</td>
</tr>
<tr>
<td>not_enforced</td>
<td>CHAR(1)</td>
<td>For system use only.</td>
</tr>
<tr>
<td>file_id</td>
<td>SMALLINT</td>
<td>DEPRECATED. This column is present in SYSVIEW, but not in the underlying system table ISYSIDX. The contents of this column are the same as dbspace_id and it is provided for compatibility. Use dbspace_id instead.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

- PRIMARY KEY (table_id, index_id)
- FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)
FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (table_id, phys_index_id) REFERENCES SYS.ISYSPHYSIDX (table_id, phys_index_id)

UNIQUE INDEX (index_name, table_id, index_category)

See also
- “SYSIDXCOL system view” on page 1366
- “SYSPHYSIDX system view” on page 1376
- “SYSDBSPACE system view” on page 1354

SYSIDXCOL system view
Each row in the SYSIDXCOL system view describes one column of an index described in the SYSIDX system view. The underlying system table for this view is ISYSIDXCOL.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>Identifies the table to which the index applies.</td>
</tr>
<tr>
<td>index_id</td>
<td>UNSIGNED INT</td>
<td>Identifies the index to which the column applies. Together, table_id and index_id identify one index described in the SYSIDX system view.</td>
</tr>
<tr>
<td>sequence</td>
<td>SMALLINT</td>
<td>Each column in an index is assigned a unique number starting at 0. The order of these numbers determines the relative significance of the columns in the index. The most important column has sequence number 0.</td>
</tr>
<tr>
<td>column_id</td>
<td>UNSIGNED INT</td>
<td>Identifies which column of the table is indexed. Together, table_id and column_id identify one column described in the SYSCOLUMN system view.</td>
</tr>
<tr>
<td>&quot;order&quot;</td>
<td>CHAR(1)</td>
<td>Indicates whether the column in the index is kept in ascending(A) or descending(D) order. This value is NULL for text indexes.</td>
</tr>
<tr>
<td>primary_column_id</td>
<td>UNSIGNED INT</td>
<td>The ID of the primary key column that corresponds to this foreign key column. The value is NULL for non foreign key columns.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

PRIMARY KEY (table_id, index_id, column_id)

FOREIGN KEY (table_id, index_id) REFERENCES SYS.ISYSPHYSIDX (table_id, index_id)

FOREIGN KEY (table_id, column_id) REFERENCES SYS.ISYSTABCOL (table_id, column_id)
SYSJAR system view

Each row in the SYSJAR system view defines a JAR file stored in the database. The underlying system table for this view is ISYSJAR.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jar_id</td>
<td>INTEGER</td>
<td>A unique number identifying the JAR file.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The internal ID for the JAR file, uniquely identifying it in the database.</td>
</tr>
<tr>
<td>creator</td>
<td>UNSIGNED INT</td>
<td>The user number of the creator of the JAR file. Can be set by the AS USER clause of the INSTALL JAVA statement.</td>
</tr>
<tr>
<td>jar_name</td>
<td>LONG VARCHAR</td>
<td>The name of the JAR file.</td>
</tr>
<tr>
<td>jar_file</td>
<td>LONG VARCHAR</td>
<td>This column is no longer used and contains NULL.</td>
</tr>
<tr>
<td>update_time</td>
<td>TIMESTAMP</td>
<td>The local time the JAR file was last updated.</td>
</tr>
<tr>
<td>update_time_utc</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>The UTC time the JAR file was last updated.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

PRIMARY KEY (jar_id)

FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

UNIQUE INDEX (jar_name)

SYSJARCOMPONENT system view

Each row in the SYSJAR system view defines a JAR file component. The underlying system table for this view is ISYSJARCOMPONENT.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component_id</td>
<td>INTEGER</td>
<td>The primary key containing the id of the component.</td>
</tr>
</tbody>
</table>
Views

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jar_id</td>
<td>INTEGER</td>
<td>A field containing the ID number of the JAR.</td>
</tr>
<tr>
<td>component_name</td>
<td>LONG VARCHAR</td>
<td>The name of the component.</td>
</tr>
<tr>
<td>component_type</td>
<td>CHAR(1)</td>
<td>This column is no longer used and contains NULL.</td>
</tr>
<tr>
<td>contents</td>
<td>LONG BINARY</td>
<td>The byte code of the JAR file.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (component_id)
- FOREIGN KEY (jar_id) REFERENCES SYS.ISYSJAR (jar_id)

See also

- “SYSJAR system view” on page 1367
- “SYSJAVACLASS system view” on page 1368

SYSJAVACLASS system view

Each row in the SYSJAVACLASS system view describes one Java class stored in the database. The underlying system table for this view is ISYSJAVACLASS.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class_id</td>
<td>INTEGER</td>
<td>The unique number for the Java class. Also the primary key for the table.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The internal ID for the Java class, uniquely identifying it in the database.</td>
</tr>
<tr>
<td>creator</td>
<td>UNSIGNED INT</td>
<td>The user number of the creator of the class. Can be set by the AS USER clause of the INSTALL JAVA statement.</td>
</tr>
<tr>
<td>jar_id</td>
<td>INTEGER</td>
<td>The id of the JAR file from which the class came.</td>
</tr>
<tr>
<td>class_name</td>
<td>LONG VARCHAR</td>
<td>The name of the Java class.</td>
</tr>
<tr>
<td>public</td>
<td>CHAR(1)</td>
<td>Indicates whether the class is public (Y) or private (N).</td>
</tr>
<tr>
<td>component_id</td>
<td>INTEGER</td>
<td>The id of the component in the SYSJARCOMPONENT system view.</td>
</tr>
<tr>
<td>update_time</td>
<td>TIMESTAMP</td>
<td>The local last update time of the class.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>update_time_utc</td>
<td>TIMESTAMP</td>
<td>The UTC last update time of the class.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (class_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
- FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
- FOREIGN KEY (component_id) REFERENCES SYS.ISYSJARCOMPONENT (component_id)

See also

- “SYSJAR system view” on page 1367
- “SYSJARCOMPONENT system view” on page 1367

**SYSLDAPSERVER system view**

The SYSLDAPSERVER system view contains one row for each LDAP server configuration object configured in the database. An LDAP server configuration object contains the configuration information necessary to connect to an LDAP server outside of SQL Anywhere. The underlying system table for this view is ISYSLDAPSERVER.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldsrv_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique identifier for the LDAP server. This ID is used by the login policy to refer to this LDAP server. This column is a foreign key to ISYSOBJECT.</td>
</tr>
<tr>
<td>ldsrv_name</td>
<td>CHAR(128)</td>
<td>The name of the LDAP server.</td>
</tr>
<tr>
<td>ldsrv_state</td>
<td>CHAR(9)</td>
<td>The state of the LDAP server. Valid values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- RESET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ACTIVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FAILED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SUSPENDED</td>
</tr>
<tr>
<td>ldsrv_start_tls</td>
<td>TINYINT</td>
<td>The valid values: 1 (ON) or 0 (OFF). If ON then LDAP over TLS is used to connect to the LDAP server. This protocol provides encrypted communication for connections and searches with the LDAP server.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ldsrv_num_retries</td>
<td>TINYINT</td>
<td>The number of attempts to authenticate with the LDAP server before returning failure or failover (if specified). Valid range: 1-60.</td>
</tr>
<tr>
<td>ldsrv_timeout</td>
<td>UNSIGNED INT</td>
<td>The timeout value for connections or searches, in milliseconds. Valid range: 1-3600000 (1 hour).</td>
</tr>
<tr>
<td>ldsrv_last_state_change</td>
<td>TIMESTAMP</td>
<td>The time when the last state change occurred. Regardless of the servers local timezone, the value is stored in Coordinated Universal Time (UTC).</td>
</tr>
<tr>
<td>ldsrv_search_url</td>
<td>CHAR(1024)</td>
<td>The LDAP URL defining the search to find the Distinguished Name (DN) for a user based on the user ID.</td>
</tr>
<tr>
<td>ldsrv_auth_url</td>
<td>CHAR(1024)</td>
<td>The LDAP search string to find the DN for a user given their user ID.</td>
</tr>
<tr>
<td>ldsrv_access_dn</td>
<td>CHAR(1024)</td>
<td>The DN used to access the LDAP server for searches to obtain DNs for other user IDs.</td>
</tr>
<tr>
<td>ldsrv_access_dn_pwd</td>
<td>VARBINARY(1024)</td>
<td>The password for the access account. The password is symmetrically encrypted when stored on disk.</td>
</tr>
</tbody>
</table>

See also

- “trusted_certificates_file option” [SQL Anywhere Server - Database Administration]

SYSLOGINMAP system view

The SYSLOGINMAP system view contains one row for each user that can connect to the database using either an integrated login, or Kerberos login. For that reason, access to this view is restricted. The underlying system table for this view is ISYSLOGINMAP.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login_mode</td>
<td>TINYINT</td>
<td>The type of login: 1 for integrated logins, 2 for Kerberos logins.</td>
</tr>
<tr>
<td>login_id</td>
<td>VARCHAR(1024)</td>
<td>Either the integrated login user profile name, or the Kerberos principal that maps to database_uid.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique identifier, one for each mapping between user ID and database user ID.</td>
</tr>
<tr>
<td>database_uid</td>
<td>UNSIGNED INT</td>
<td>The database user ID to which the login ID is mapped.</td>
</tr>
</tbody>
</table>
Constraints on underlying system table

<table>
<thead>
<tr>
<th>PRIMARY KEY (login_mode, login_id)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL</td>
</tr>
<tr>
<td>FOREIGN KEY (database_uid) REFERENCES SYS.ISYSUSER (user_id)</td>
</tr>
</tbody>
</table>

SYSLOGINPOLICY system view

The underlying system table for this view is ISYSLOGINPOLICY.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login_policy_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique identifier for the login policy.</td>
</tr>
<tr>
<td>login_policy_name</td>
<td>CHAR(128)</td>
<td>The name of the login policy.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

<table>
<thead>
<tr>
<th>PRIMARY KEY (login_policy_id)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOREIGN KEY (login_policy_id) REFERENCES SYS.ISYSOBJECT (object_id)</td>
</tr>
<tr>
<td>UNIQUE INDEX (login_policy_name)</td>
</tr>
</tbody>
</table>

See also

- “SYSLOGINPOLICYOPTION system view” on page 1371
- “SYSUSER system view” on page 1412

SYSLOGINPOLICYOPTION system view

The underlying system table for this view is ISYSLOGINPOLICYOPTION.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login_policy_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique identifier for the login policy.</td>
</tr>
<tr>
<td>login_option_name</td>
<td>CHAR(128)</td>
<td>The name of the login policy.</td>
</tr>
<tr>
<td>login_option_value</td>
<td>LONG VARCHAR</td>
<td>The value of the login policy at the time it was created.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

<table>
<thead>
<tr>
<th>PRIMARY KEY (login_policy_id, login_option_name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOREIGN KEY (login_policy_id) REFERENCES SYS.ISYSLOGINPOLICY (login_policy_id)</td>
</tr>
</tbody>
</table>
SYSMIRROROPTION system view

The underlying system table for this view is ISYSMIRROROPTION.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option_name</td>
<td>CHAR(128)</td>
<td>The name of the option.</td>
</tr>
<tr>
<td>option_value</td>
<td>LONG VARCHAR</td>
<td>The value of the option when the mirror was created.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Values in this column are hidden from users that do not have the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SELECT ANY TABLE system privilege.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

PRIMARY KEY (option_name)

See also
- “SYSMIRRORORSERVER system view” on page 1372
- “SYSMIRRORORSERVEROPTION system view” on page 1373
- “SET MIRROR OPTION statement” on page 968

SYSMIRRORORSERVER system view

The underlying system table for this view is ISYSMIRRORORSERVER.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique identifier for the mirror server.</td>
</tr>
<tr>
<td>server_name</td>
<td>CHAR(128)</td>
<td>The name of the server.</td>
</tr>
<tr>
<td>server_type</td>
<td>CHAR(20)</td>
<td>The type of server. The value can be one of PRIMARY, MIRROR, ARBITER, PARTNER, or COPY.</td>
</tr>
<tr>
<td>parent</td>
<td>UNSIGNED BIGINT</td>
<td>The parent server. If the value is NULL, then the server is the primary or mirror server in a database mirroring system. If there is a value in this column, it is the ID of the server that is the parent of the current server.</td>
</tr>
</tbody>
</table>
## alternate_parent column

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alternate_parent</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the server that is used as an alternate parent if the current parent becomes unavailable.</td>
</tr>
</tbody>
</table>

### Constraints on underlying system table

- PRIMARY KEY (object_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id)
- UNIQUE INDEX (server_name)

### See also

- “SYSMIRROROPTION system view” on page 1372
- “SYSMIRRORSERVEROPTION system view” on page 1373
- “Copy node parent determination” [SQL Anywhere Server - Database Administration]
- “CREATE MIRROR SERVER statement” on page 615

## SYSMIRRORSERVEROPTION system view

The underlying system table for this view is ISYSMIRRORSERVEROPTION.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_id</td>
<td>UNSIGNED BIGINT</td>
<td>A unique identifier for the mirror server.</td>
</tr>
<tr>
<td>option_name</td>
<td>CHAR(128)</td>
<td>The name of the option.</td>
</tr>
<tr>
<td>option_value</td>
<td>LONG VARCHAR</td>
<td>The value of the option when the mirror was created.</td>
</tr>
</tbody>
</table>

### Constraints on underlying system table

- PRIMARY KEY (server_id, option_name)
- FOREIGN KEY (server_id) references SYS.ISYSMIRRORSERVER (object_id)

### See also

- “SYSMIRROROPTION system view” on page 1372
- “SYSMIRRORSERVER system view” on page 1372

## SYSMVOPTION system view

Each row in the SYSMVOPTION system view describes the setting of one option value for a materialized view or text index at the time of its creation. The name of the option can be found in the SYSMVOPTIONNAME system view. The underlying system table for this view is ISYSMVOPTION.
Views

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>view_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the materialized view.</td>
</tr>
<tr>
<td>option_id</td>
<td>UNSIGNED INT</td>
<td>A unique number identifying the option in the database. To see the option name, see the SYSMVOPTIONNAME system view.</td>
</tr>
<tr>
<td>option_value</td>
<td>LONG VARCHAR</td>
<td>The value of the option when the materialized view was created.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (view_object_id, option_id)
- FOREIGN KEY (view_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
- FOREIGN KEY (option_id) REFERENCES SYS.ISYSMVOPTIONNAME (option_id)

See also
- “SYSMVOPTIONNAME system view” on page 1374
- “Viewing text index terms and settings (Sybase Central)” [SQL Anywhere Server - SQL Usage]
- “Advanced: Viewing materialized view information in the catalog” [SQL Anywhere Server - SQL Usage]

SYSMVOPTIONNAME system view

Each row in the SYSMVOPTION system view gives the name option value for a materialized view or text index at the time of its creation. The value for the option can be found in the SYSMVOPTION system view. The underlying system table for this view is ISYSMVOPTIONNAME.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option_id</td>
<td>UNSIGNED INT</td>
<td>A number uniquely identifying the option in the database.</td>
</tr>
<tr>
<td>option_name</td>
<td>CHAR(128)</td>
<td>The name of the option.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (option_id)
- UNIQUE INDEX (option_name)

See also
- “SYSMVOPTION system view” on page 1373
- “Viewing text index terms and settings (Sybase Central)” [SQL Anywhere Server - SQL Usage]
- “Advanced: Viewing materialized view information in the catalog” [SQL Anywhere Server - SQL Usage]
SYSOBJECT system view

Each row in the SYSOBJECT system view describes a database object. The underlying system table for this view is ISYSOBJECT.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The internal ID for the object, uniquely identifying it in the database.</td>
</tr>
<tr>
<td>status</td>
<td>TINYINT</td>
<td>The status of the object. Values include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 <strong>(valid)</strong>  The object is available for use by the database server. This status is synonymous with ENABLED. That is, if you ENABLE an object, the status changes to VALID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 <strong>(invalid)</strong> An attempt to recompile the object after an internal operation has failed, for example, after a schema-altering modification to an object on which it depends. The database server continues to try to recompile the object whenever it is referenced in a statement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 4 <strong>(disabled)</strong> The object has been explicitly disabled by the user, for example using an ALTER TABLE...DISABLE VIEW DEPENDENCIES statement.</td>
</tr>
<tr>
<td>object_type</td>
<td>TINYINT</td>
<td>Type of object.</td>
</tr>
<tr>
<td>creation_time</td>
<td>TIME-STAMP</td>
<td>The local date and time when the object was created.</td>
</tr>
<tr>
<td>object_type_str</td>
<td>CHAR (128)</td>
<td>Type of object.</td>
</tr>
<tr>
<td>creation_time_utc</td>
<td>TIME-STAMP WITH TIME ZONE</td>
<td>The UTC date and time when the object was created.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

PRIMARY KEY (object_id)

SYSOPTION system view

The SYSOPTION system view contains the options one row for each option setting stored in the database. Each user can have their own setting for a given option. In addition, settings for the PUBLIC role define the default settings to be used for users that do not have their own setting. The underlying system table for this view is ISYSOPTION.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>UNSIGNED INT</td>
<td>The user number to whom the option setting applies.</td>
</tr>
<tr>
<td>&quot;option&quot;</td>
<td>CHAR(128)</td>
<td>The name of the option.</td>
</tr>
<tr>
<td>&quot;setting&quot;</td>
<td>LONG VARCHAR</td>
<td>The current setting for the option.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (user_id, "option")
- FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)

**SYSOPTSTAT system view**

The SYSOPTSTAT system view stores the cost model calibration information as computed by the ALTER DATABASE CALIBRATE statement. The contents of this view are for internal use only and are best accessed via the sa_get_dtt system procedure. The underlying system table for this view is ISYSOPTSTAT.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat_id</td>
<td>UNSIGNED INT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>group_id</td>
<td>UNSIGNED INT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>format_id</td>
<td>SMALLINT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>data</td>
<td>LONG BINARY</td>
<td>For system use only.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (stat_id, group_id, format_id)

**SYSPHYSIDX system view**

Each row in the SYSPHYSIDX system view defines a physical index in the database. The underlying system table for this view is ISYSPHYSIDX.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>The object ID of the table to which the index corresponds.</td>
</tr>
<tr>
<td>phys_index_id</td>
<td>UNSIGNED INT</td>
<td>The unique number of the physical index within its table.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>root</td>
<td>INTEGER</td>
<td>Identifies the location of the root page of the physical index in the database file.</td>
</tr>
<tr>
<td>key_value_count</td>
<td>UNSIGNED INT</td>
<td>The number of distinct key values in the index.</td>
</tr>
<tr>
<td>leaf_page_count</td>
<td>UNSIGNED INT</td>
<td>The number of leaf index pages.</td>
</tr>
<tr>
<td>depth</td>
<td>UNSIGNED SMALL-INT</td>
<td>The depth (number of levels) of the physical index.</td>
</tr>
<tr>
<td>max_key_distance</td>
<td>UNSIGNED INT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>seq_transitions</td>
<td>UNSIGNED INT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>rand_transitions</td>
<td>UNSIGNED INT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>rand_distance</td>
<td>UNSIGNED INT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>allocation_bitmap</td>
<td>LONG VARBIT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>long_value_bitmap</td>
<td>LONG VARBIT</td>
<td>For system use only.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

PRIMARY KEY (table_id, phys_index_id)

**See also**

- “SYSIDX system view” on page 1364
- “SYSIDXCOL system view” on page 1366

**SYSPROCEDURE system view**

Each row in the SYSPROCEDURE system view describes one procedure in the database. The underlying system table for this view is ISYSPROCEDURE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc_id</td>
<td>UNSIGNED INT</td>
<td>Each procedure is assigned a unique number (the procedure number).</td>
</tr>
<tr>
<td>creator</td>
<td>UNSIGNED INT</td>
<td>The owner of the procedure.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The internal ID for the procedure, uniquely identifying it in the database.</td>
</tr>
</tbody>
</table>
**Column name** | **Data type** | **Description**
--- | --- | ---
proc_name | CHAR(128) | The name of the procedure. One creator cannot have two procedures with the same name.
proc_defn | LONG VARCHAR | The definition of the procedure.
remarks | LONG VARCHAR | Remarks about the procedure. This value is stored in the ISYSREMARK system table.
replicate | CHAR(1) | This property is for internal use only.
srvid | UNSIGNED INT | If the procedure is a proxy for a procedure on a remote database server, indicates the remote server.
source | LONG VARCHAR | The preserved source for the procedure. This value is stored in the ISYSSOURCE system table.
avg_num_rows | FLOAT | Information collected for use in query optimization when the procedure appears in the FROM clause.
avg_cost | FLOAT | Information collected for use in query optimization when the procedure appears in the FROM clause.
stats | LONG BINARY | Information collected for use in query optimization when the procedure appears in the FROM clause.

**Constraints on underlying system table**

- PRIMARY KEY (proc_id)
- FOREIGN Key (srvid) REFERENCES SYS.ISYSSERVER (srvid)
- FOREIGN Key (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
- FOREIGN Key (creator) REFERENCES SYS.ISYSUSER (user_id)
- UNIQUE INDEX (proc_name, creator)

**SYSPROCPARM system view**

Each row in the SYSPROCPARM system view describes one parameter to a procedure in the database. The underlying system table for this view is ISYSPROCPARM.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc_id</td>
<td>UNSIGNED INT</td>
<td>Uniquely identifies the procedure to which the parameter belongs.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>parm_id</td>
<td>SMALLINT</td>
<td>Each procedure starts numbering parameters at 1. The order of parameter numbers corresponds to the order in which they were defined. For functions, the first parameter has the name of the function and represents the return value for the function.</td>
</tr>
<tr>
<td>parm_type</td>
<td>SMALLINT</td>
<td>The type of parameter is one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 0  Normal parameter (variable)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1  Result variable - used with a procedure that returns result sets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2  SQLSTATE error value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 3  SQLCODE error value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 4  Return value from function</td>
</tr>
<tr>
<td>parm_mode_in</td>
<td>CHAR(1)</td>
<td>Indicates whether the parameter supplies a value to the procedure (IN or INOUT parameters).</td>
</tr>
<tr>
<td>parm_mode_out</td>
<td>CHAR(1)</td>
<td>Indicates whether the parameter returns a value from the procedure (OUT or INOUT parameters) or columns in the RESULT clause.</td>
</tr>
<tr>
<td>domain_id</td>
<td>SMALLINT</td>
<td>Identifies the data type for the parameter, by the data type number listed in the SYSDOMAIN system view.</td>
</tr>
<tr>
<td>width</td>
<td>BIGINT</td>
<td>Contains the length of a string parameter, the precision of a numeric parameter, or the number of bytes of storage for any other data type.</td>
</tr>
<tr>
<td>scale</td>
<td>SMALLINT</td>
<td>For numeric data types, the number of digits after the decimal point. For all other data types, the value of this column is 1.</td>
</tr>
<tr>
<td>user_type</td>
<td>SMALLINT</td>
<td>The user type of the parameter, if applicable.</td>
</tr>
<tr>
<td>parm_name</td>
<td>CHAR(128)</td>
<td>The name of the procedure parameter.</td>
</tr>
<tr>
<td>&quot;default&quot;</td>
<td>LONG VARCHAR</td>
<td>Default value of the parameter. Provided for informational purposes only.</td>
</tr>
<tr>
<td>remarks</td>
<td>LONG VARCHAR</td>
<td>Always returns NULL. Provided to allow the use of previous versions of ODBC drivers with newer personal database servers.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>base_type_str</td>
<td>VAR-CHAR(32767)</td>
<td>The annotated type string representing the physical type of the parameter.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**
- PRIMARY KEY (proc_id, parm_id)
- FOREIGN KEY (proc_id) REFERENCES SYS.ISYSPROCEDURE (proc_id)
- FOREIGN KEY (domain_id) REFERENCES SYS.ISYSDOMAIN (domain_id)
- FOREIGN KEY (user_type) REFERENCES SYS.ISYSUSERTYPE (type_id)

**SYSPROCPERM system view**
Each row of the SYSPROCPERM system view describes a user who has been granted EXECUTE privilege on a procedure. The underlying system table for this view is ISYSPROCPERM.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc_id</td>
<td>UNSIGNED INT</td>
<td>The procedure number uniquely identifies the procedure for which EXECUTE privilege has been granted.</td>
</tr>
<tr>
<td>grantee</td>
<td>UNSIGNED INT</td>
<td>The user number of the privilege grantee.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**
- PRIMARY KEY (proc_id, grantee)
- FOREIGN KEY (grantee) REFERENCES SYS.ISYSUSER (user_id)
- FOREIGN KEY (proc_id) REFERENCES SYS.ISYSPROCEDURE (proc_id)

**SYSPROXYTAB system view**
Each row of the SYSPROXYTAB system view describes the remote parameters of one proxy table. The underlying system table for this view is ISYSPROXYTAB.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the proxy table.</td>
</tr>
<tr>
<td>existing_obj</td>
<td>CHAR(1)</td>
<td>Indicates whether the proxy table previously existed on the remote server.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>srvid</td>
<td>UNSIGNED INT</td>
<td>The unique ID for the remote server associated with the proxy table.</td>
</tr>
<tr>
<td>remote_location</td>
<td>LONG VARCHAR</td>
<td>The location of the proxy table on the remote server.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

- PRIMARY KEY (table_object_id)
- FOREIGN KEY (table_object_id) REFERENCES ISYSOBJECT (object_id) MATCH UNIQUE FULL
- FOREIGN KEY (srvid) REFERENCES SYS.ISYSSERVER (srvid)

**SYSPUBLICATION system view**

Each row in the SYSPUBLICATION system view describes a SQL Remote or MobiLink publication. The underlying system table for this view is ISYSPUBLICATION.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>UNSIGNED INT</td>
<td>A number uniquely identifying the publication.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The internal ID for the publication, uniquely identifying it in the database.</td>
</tr>
<tr>
<td>creator</td>
<td>UNSIGNED INT</td>
<td>The owner of the publication.</td>
</tr>
<tr>
<td>publication_name</td>
<td>CHAR(128)</td>
<td>The name of the publication.</td>
</tr>
<tr>
<td>remarks</td>
<td>LONG VARCHAR</td>
<td>Remarks about the publication. This value is stored in the ISYSREMARK system table.</td>
</tr>
<tr>
<td>type</td>
<td>CHAR(1)</td>
<td>This column is deprecated.</td>
</tr>
</tbody>
</table>
## sync_type

**Data type:** UNSIGNED INT

**Description:** The type of synchronization for the publication. Values include:

- **0 (logscan)**: This is a regular publication that uses the transaction log to upload all relevant data that has changed since the last upload.

- **1 (scripted upload)**: For this publication, the transaction log is ignored and the upload is defined by the user using stored procedures. Information about the stored procedures is stored in the ISYSSYNCSHIFT system table.

- **2 (download only)**: This is a download-only publication; no data is uploaded.

### Constraints on underlying system table

- PRIMARY KEY (publication_id)

- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

- FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)

- UNIQUE INDEX (publication_name, creator)

### See also

- “Scripted upload” [MobiLink - Client Administration]
- “SYSSYNCSHIFT system view” on page 1398

## SYSREMARK system view

Each row in the SYSREMARK system view describes a remark (or comment) for an object. The underlying system table for this view is ISYSREMARK.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The internal ID for the object that has an associated remark.</td>
</tr>
<tr>
<td>remarks</td>
<td>LONG VARCHAR</td>
<td>The remark or comment associated with the object.</td>
</tr>
</tbody>
</table>

### Constraints on underlying system table

- PRIMARY KEY (object_id)

- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
SYSREMOOTEOPTION system view

Each row in the SYSREMOOTEOPTION system view describes the value of a SQL Remote message link parameter. The underlying system table for this view is ISYSREMOOTEOPTION.

Some columns in this view contain potentially sensitive data. The SYSREMOOTEOPTION2 view provides public access to the data in this view except for the potentially sensitive columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option_id</td>
<td>UNSIGNED INT</td>
<td>An identification number for the message link parameter.</td>
</tr>
<tr>
<td>user_id</td>
<td>UNSIGNED INT</td>
<td>The user ID for which the parameter is set.</td>
</tr>
<tr>
<td>&quot;setting&quot;</td>
<td>VARCHAR(255)</td>
<td>The value of the message link parameter.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

PRIMAR Y KEY (option_id, user_id)

FOREIGN KEY (option_id) REFERENCES SYS.ISYSREMOOTEOPTIONTYPE (option_id)

FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)

SYSREMOOTEOPTIONTYPE system view

Each row in the SYSREMOOTEOPTIONTYPE system view describes one of the SQL Remote message link parameters. The underlying system table for this view is ISYSREMOOTEOPTIONTYPE.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option_id</td>
<td>UNSIGNED INT</td>
<td>An identification number for the message link parameter.</td>
</tr>
<tr>
<td>type_id</td>
<td>SMALLINT</td>
<td>An identification number for the message type that uses the parameter.</td>
</tr>
<tr>
<td>option</td>
<td>VARCHAR(128)</td>
<td>The name of the message link parameter.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

PRIMAR Y KEY (option_id)

FOREIGN KEY (type_id) REFERENCES SYS.ISYSREMOOTETYPE (type_id)

SYSREMOOTETYPE system view

The SYSREMOOTETYPE system view contains information about SQL Remote. The underlying system table for this view is ISYSREMOOTETYPE.
Column name | Data type | Description
--- | --- | ---
type_id | SMALLINT | Identifies which of the message systems supported by SQL Remote is to be used to send messages to the user.
object_id | UNSIGNED BIGINT | The internal ID for the remote type, uniquely identifying it in the database.
type_name | CHAR(128) | The name of the message system supported by SQL Remote.
publisher_address | LONG VARCHAR | The address of the remote database publisher.
remarks | LONG VARCHAR | Remarks about the remote type. This value is stored in the ISYSREMARK system table.

Constraints on underlying system table

- PRIMARY KEY (type_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
- UNIQUE INDEX (type_name)

SYSREMOTEUSER system view

Each row in the SYSREMOTEUSER system view describes a user ID with the REMOTE system privilege (a subscriber), together with the status of SQL Remote messages that were sent to and from that user. The underlying system table for this view is ISYSREMOTEUSER.

Column name | Data type | Description
--- | --- | ---
user_id | UNSIGNED INT | The user number of the user with REMOTE privilege.
consolidate | CHAR(1) | Indicates whether the user was granted CONSOLIDATE privilege (Y) or REMOTE privileges (N).
type_id | SMALLINT | Identifies which of the message systems supported by SQL Remote is used to send messages to the user.
address | LONG VARCHAR | The address to which SQL Remote messages are to be sent. The address must be appropriate for the address_type.
frequency | CHAR(1) | How frequently SQL Remote messages are sent.
send_time | TIME | The next time messages are to be sent to this user.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log_send</td>
<td>UNSIGNED BIGINT</td>
<td>Messages are sent only to subscribers for whom log_send is greater than log_sent.</td>
</tr>
<tr>
<td>time_sent</td>
<td>TIMESTAMP</td>
<td>The local time the most recent message was sent to this subscriber.</td>
</tr>
<tr>
<td>log_sent</td>
<td>UNSIGNED BIGINT</td>
<td>The log offset for the most recently sent operation.</td>
</tr>
<tr>
<td>confirm_sent</td>
<td>UNSIGNED BIGINT</td>
<td>The log offset for the most recently confirmed operation from this subscriber.</td>
</tr>
<tr>
<td>send_count</td>
<td>INTEGER</td>
<td>How many SQL Remote messages have been sent.</td>
</tr>
<tr>
<td>resend_count</td>
<td>INTEGER</td>
<td>Counter to ensure that messages are applied only once at the subscriber database.</td>
</tr>
<tr>
<td>time_received</td>
<td>TIMESTAMP</td>
<td>The local time when the most recent message was received from this subscriber.</td>
</tr>
<tr>
<td>log_received</td>
<td>UNSIGNED BIGINT</td>
<td>The log offset in the database of the subscriber for the operation that was most recently received at the current database.</td>
</tr>
<tr>
<td>confirm_received</td>
<td>UNSIGNED BIGINT</td>
<td>The log offset in the database of the subscriber for the most recent operation for which a confirmation message has been sent.</td>
</tr>
<tr>
<td>receive_count</td>
<td>INTEGER</td>
<td>How many messages have been received.</td>
</tr>
<tr>
<td>rereceive_count</td>
<td>INTEGER</td>
<td>Counter to ensure that messages are applied only once at the current database.</td>
</tr>
<tr>
<td>time_sent_utc</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>The UTC time the most recent message was sent to this subscriber.</td>
</tr>
<tr>
<td>time_received_utc</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>The UTC time when the most recent message was received from this subscriber.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

- PRIMARY KEY (user_id)
- FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)
- FOREIGN KEY (type_id) REFERENCES SYS.ISYSREMTYPE (type_id)
- UNIQUE INDEX (type_id, address)
SYSROLEGRANT system view

The SYSROLEGRANT system view stores information about role membership and type of membership. The underlying system table for this view is ISYSROLEGRANT.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grant_id</td>
<td>UNSIGNED INT</td>
<td>ID used to identify each GRANT statement.</td>
</tr>
<tr>
<td>role_id</td>
<td>UNSIGNED INT</td>
<td>ID of the role being granted, as per ISYSUSER.</td>
</tr>
<tr>
<td>grantee</td>
<td>UNSIGNED INT</td>
<td>ID of the user being granted the role, as per ISYSUSER.</td>
</tr>
<tr>
<td>grant_type</td>
<td>TINYINT</td>
<td>Describes type of grant using 3 digits. The first bit from the right is whether privilege has been granted. The second digit is whether administration rights have been given. The third digit is whether system privileges are inheritable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 001 Privilege granted, with no inheritance, and no administration rights. Applicable only for legacy non-inheritable authorities except DBA and REMOTE DBA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 101 Privilege granted, with inheritance, but no administration rights.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 110 Only administration rights have been granted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 111 Privilege granted, with inheritance, and with administration rights.</td>
</tr>
<tr>
<td>grant_scope</td>
<td>TINYINT</td>
<td>Used by SET USER and CHANGE PASSWORD to set the scope of the grant. Values can be one or more of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1 ANY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 2 User list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 4 Role list</td>
</tr>
<tr>
<td>grantor</td>
<td>CHAR(128)</td>
<td>The name of the grantor.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

PRIMARY KEY (grant_id)

FOREIGN KEY (role_id) REFERENCES sys.isysuser (user_id)

FOREIGN KEY (grantee) REFERENCES sys.isysuser (user_id)

UNIQUE INDEX (role_id, grantee, grant_scope)
SYSROLEGRANTEXT system view

When you grant the SET USER and the CHANGE PASSWORD system privileges, you can specify a list of users or roles that the grantee can grant them to. The SYSROLEGRANTEXT system view stores information about which users and roles the grantee can grant SET USER and CHANGE PASSWORD system privileges to.

The underlying system table for this view is ISYSROLEGRANTEXT.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grant_id</td>
<td>UNSIGNED INT</td>
<td>ID used to identify each GRANT statement.</td>
</tr>
<tr>
<td>user_id</td>
<td>UNSIGNED INT</td>
<td>The user_ids specified in user-list or role-list in a particular extended grant</td>
</tr>
</tbody>
</table>

Remarks

When you grant or revoke the SET USER or CHANGE PASSWORD privilege, either with the user-list option or with ANY WITH ROLES role-list option, this view is updated with the values from the extended syntax.

Constraints on underlying system table

- PRIMARY KEY (grant_id, user_id)
- FKEY( grant_id ) REFERENCES ISYSROLEGRANT( grant_id )
- FKEY( user_id ) REFERENCES ISYSUSER( user_id )

SYSROLEGRANTS consolidated view

The SYSROLEGRANTS system view stores information about role membership and type of membership, just like the SYSROLEGRANT system view does. However, SYSROLEGRANTS includes role names and grantee names (not just IDs). The underlying system tables for this view are ISYSROLEGRANT and ISYSUSER.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grant_id</td>
<td>UNSIGNED INT</td>
<td>ID used to identify each GRANT statement.</td>
</tr>
<tr>
<td>role_id</td>
<td>UNSIGNED INT</td>
<td>ID of the role being granted, as per ISYSUSER.</td>
</tr>
<tr>
<td>role_name</td>
<td>CHAR(128)</td>
<td>The name of the role.</td>
</tr>
<tr>
<td>grantee</td>
<td>UNSIGNED INT</td>
<td>ID of the user being granted the role, as per ISYSUSER.</td>
</tr>
<tr>
<td>grantee_name</td>
<td>CHAR(128)</td>
<td>The name of the grantee.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>grant_type</td>
<td>TINYINT</td>
<td>Describes type of grant using 3 bits. The first bit from the right is whether privilege has been granted. The second digit is whether administration rights have been given. The third digit is whether system privileges are inheritable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 001 Privilege granted, with no inheritance, and no administration rights. Applicable only for legacy non-inheritable authorities except DBA and REMOTE DBA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 101 Privilege granted, with inheritance, but no administration rights.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 110 Only administration rights have been granted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 111 Privilege granted, with inheritance, and with administration rights.</td>
</tr>
<tr>
<td>grant_scope</td>
<td>TINYINT</td>
<td>Used by SET USER and CHANGE PASSWORD to set the scope of the grant. Values can be one or more of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1 ANY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 2 User list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 4 Role list</td>
</tr>
<tr>
<td>grantor</td>
<td>CHAR(128)</td>
<td>The name of the grantor.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

- PRIMARY KEY (grant_id)
- FOREIGN KEY (role_id) REFERENCES SYS.ISYSUSER (user_id)
- FOREIGN KEY (grantee) REFERENCES SYS.ISYSUSER (user_id)
- INDEX (role_id, grantee, grant_scope)

**SYSSCHEDULE system view**

Each row in the SYSSCHEDULE system view describes a time at which an event is to fire, as specified by the SCHEDULE clause of CREATE EVENT. The underlying system table for this view is ISYSSCHEDULE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_id</td>
<td>UNSIGNED INT</td>
<td>The unique number assigned to each event.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sched_name</td>
<td>VARCHAR(128)</td>
<td>The name associated with the schedule for the event.</td>
</tr>
<tr>
<td>recurring</td>
<td>TINYINT</td>
<td>Indicates if the schedule is repeating.</td>
</tr>
<tr>
<td>start_time</td>
<td>TIME</td>
<td>The schedule start time.</td>
</tr>
<tr>
<td>stop_time</td>
<td>TIME</td>
<td>The schedule stop time if BETWEEN was used.</td>
</tr>
<tr>
<td>start_date</td>
<td>DATE</td>
<td>The first date on which the event is scheduled to execute.</td>
</tr>
<tr>
<td>days_of_week</td>
<td>TINYINT</td>
<td>A bit mask indicating the days of the week on which the event is scheduled:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x01 = Sunday</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x02 = Monday</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x04 = Tuesday</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x08 = Wednesday</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x10 = Thursday</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x20 = Friday</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x40 = Saturday</td>
</tr>
<tr>
<td>days_of_month</td>
<td>UNSIGNED INT</td>
<td>A bit mask indicating the days of the month on which the event is scheduled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x01 = first day</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x02 = second day</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x40000000 = 31st day</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● x80000000 = last day of month</td>
</tr>
<tr>
<td>interval_units</td>
<td>CHAR(10)</td>
<td>The interval unit specified by EVERY:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● HH = hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● NN = minutes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● SS = seconds</td>
</tr>
<tr>
<td>interval_amt</td>
<td>INTEGER</td>
<td>The period specified by EVERY.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

- PRIMARY KEY (event_id, sched_name)
- FOREIGN KEY (event_id) REFERENCES SYS.ISYSEVENT (event_id)

**SYSSEQUENCE system view**

The SYSSEQUENCE system view contains one row for each user-defined sequence. The underlying system table for this view is ISYSSEQUENCE.
### Constraints on underlying system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The unique number assigned to each sequence.</td>
</tr>
<tr>
<td>owner</td>
<td>UNSIGNED INT</td>
<td>The owner of the sequence.</td>
</tr>
<tr>
<td>min_value</td>
<td>BIGINT</td>
<td>The minimum value allowed for the sequence.</td>
</tr>
<tr>
<td>max_value</td>
<td>BIGINT</td>
<td>The maximum value allowed for the sequence.</td>
</tr>
<tr>
<td>increment_by</td>
<td>BIGINT</td>
<td>The increment value for the sequence.</td>
</tr>
<tr>
<td>start_with</td>
<td>BIGINT</td>
<td>The starting value for the sequence.</td>
</tr>
<tr>
<td>cache</td>
<td>UNSIGNED INT</td>
<td>The number of sequence values to preallocate in memory for faster access. A value of 0 indicates that values are not to be preallocated</td>
</tr>
<tr>
<td>cycle</td>
<td>TINYINT</td>
<td>Whether values should continue to be generated after the maximum or minimum value is reached.</td>
</tr>
<tr>
<td>resume_at</td>
<td>BIGINT</td>
<td>The RESTART WITH value specified by the ALTER SEQUENCE statement. The value is NULL if no ALTER RE-START WITH statement has been executed.</td>
</tr>
<tr>
<td>sequence_name</td>
<td>CHAR(128)</td>
<td>The name of the sequence.</td>
</tr>
</tbody>
</table>

### SYSSEQUENCEPERM system view

The SYSSEQUENCEPERM system view records the privileges that users or groups hold on sequences. The underlying system table for this view is ISYSSEQUENCEPERM.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence_id</td>
<td>UNSIGNED BIGINT</td>
<td>The unique number assigned to each sequence.</td>
</tr>
<tr>
<td>grantee</td>
<td>UNSIGNED INT</td>
<td>The ID of the user or group with privileges to alter or drop the sequence.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>grantor</td>
<td>UNSIGNED INT</td>
<td>The ID of the user who granted the privileges for the sequence.</td>
</tr>
<tr>
<td>privilege_type</td>
<td>SMALLINT</td>
<td>The type of privileges granted to the user or group on the sequence.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- **PRIMARY KEY** (sequence_id, grantee, privilege_type)
- **FOREIGN KEY** (sequence_id) REFERENCES SYS.ISYSEQUENCE (object_id)
- **FOREIGN KEY** (grantee) REFERENCES SYS.ISYSUSER (user_id)
- **FOREIGN KEY** (grantor) REFERENCES SYS.ISYSUSER (user_id)

**SYSSERVER system view**

Each row in the SYSSERVER system view describes a remote server. The underlying system table for this view is ISYSSERVER.

**Note**

Previous versions of the catalog contained a SYSSERVERS system table. That table has been renamed to be ISYSSERVER (without an 'S'), and is the underlying table for this view.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srvid</td>
<td>UNSIGNED INT</td>
<td>An identifier for the remote server.</td>
</tr>
<tr>
<td>srvname</td>
<td>VARCHAR(128)</td>
<td>The name of the remote server.</td>
</tr>
<tr>
<td>srvclass</td>
<td>LONG VARCHAR</td>
<td>The server class, as specified in the CREATE SERVER statement.</td>
</tr>
<tr>
<td>srvinfo</td>
<td>LONG VARCHAR</td>
<td>Server information.</td>
</tr>
<tr>
<td>srvreadonly</td>
<td>CHAR(1)</td>
<td>Whether the server is read-only.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- **PRIMARY KEY** (srvid)

**SYSSOURCE system view**

Each row in the SYSSOURCE system view contains the source code, if applicable, for an object listed in the SYSOBJECT system view. The underlying system table for this view is ISYSSOURCE.
**Column name** | **Data type** | **Description**
---|---|---
object_id | UNSIGNED BIGINT | The internal ID for the object whose source code is being defined.
source | LONG VARCHAR | This column contains the original source code for the object if the preserve_source_format database option is On when the object was created.

**Constraints on underlying system table**

- PRIMARY KEY (object_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

**See also**

- “preserve_source_format option” [SQL Anywhere Server - Database Administration]

**SYSSPATIALREFERENCESYSTEM system view**

Each row of the SYSSPATIALREFERENCESYSTEM system view describes an SRS defined in the database. The underlying system table for this view is ISYSSPATIALREFERENCESYSTEM.

This view offers slightly different amount of information than the ST_SPATIAL_REFERENCE_SYSTEMS system view.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>owner</td>
<td>UNSIGNED INT</td>
<td>The owner of the SRS.</td>
</tr>
<tr>
<td>srs_name</td>
<td>CHAR(128)</td>
<td>The name of the SRS.</td>
</tr>
<tr>
<td>srs_id</td>
<td>INTEGER</td>
<td>The numeric identifier (SRID) for the spatial reference system.</td>
</tr>
<tr>
<td>round_earth</td>
<td>CHAR(1)</td>
<td>Whether the SRS type is ROUND EARTH (Y) or PLANAR (N).</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>axis_order</td>
<td>CHAR(12)</td>
<td>Describes how the database server interprets points with regards to latitude and longitude (for example when using the ST_Lat and ST_Long methods). For non-geographic spatial reference systems, the axis order is x/y/z/m. For geographic spatial reference systems, the default axis order is long/lat/z/m; lat/long/z/m is also supported.</td>
</tr>
<tr>
<td>snap_to_grid</td>
<td>DOUBLE</td>
<td>Defines the size of the grid SQL Anywhere uses when performing calculations.</td>
</tr>
<tr>
<td>tolerance</td>
<td>DOUBLE</td>
<td>Defines the precision to use when comparing points.</td>
</tr>
<tr>
<td>semi_major_axis</td>
<td>DOUBLE</td>
<td>Distance from center of the ellipsoid to the equator for a ROUND EARTH SRS.</td>
</tr>
<tr>
<td>semi_minor_axis</td>
<td>DOUBLE</td>
<td>Distance from center of the ellipsoid to the poles for a ROUND EARTH SRS.</td>
</tr>
<tr>
<td>inv_flattening</td>
<td>DOUBLE</td>
<td>The inverse flattening used for the ellipsoid in a ROUND EARTH SRS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inverse flattening (f) is a mathematical value that defines the degree of squashing of the pole of a spheroid towards its equator. The value ranges from no flattening (a perfect circle) to complete flattening (a straight line). Inverse flattening is the value of 1/f, as follows: 1/f = (semi_major_axis) / (semi_major_axis - semi_minor_axis)</td>
</tr>
<tr>
<td>min_x</td>
<td>DOUBLE</td>
<td>The minimum x value allowed in coordinates.</td>
</tr>
<tr>
<td>max_x</td>
<td>DOUBLE</td>
<td>The maximum x value allowed in coordinates.</td>
</tr>
<tr>
<td>min_y</td>
<td>DOUBLE</td>
<td>The minimum y value allowed in coordinates.</td>
</tr>
<tr>
<td>max_y</td>
<td>DOUBLE</td>
<td>The maximum y value allowed in coordinates.</td>
</tr>
<tr>
<td>min_z</td>
<td>DOUBLE</td>
<td>The minimum z value allowed in coordinates.</td>
</tr>
<tr>
<td>max_z</td>
<td>DOUBLE</td>
<td>The maximum z value allowed in coordinates.</td>
</tr>
<tr>
<td>min_m</td>
<td>DOUBLE</td>
<td>The minimum m value allowed in coordinates.</td>
</tr>
<tr>
<td>max_m</td>
<td>DOUBLE</td>
<td>The maximum m value allowed in coordinates.</td>
</tr>
<tr>
<td>organization</td>
<td>LONG Var-CHAR</td>
<td>The name of the organization that created the coordinate system used by the spatial reference system.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>organization_coord-sys_id</td>
<td>INTEGER</td>
<td>The ID given to the coordinate system by the organization that created it.</td>
</tr>
<tr>
<td>srs_type</td>
<td>CHAR(11)</td>
<td>The type of SRS as defined by the SQL/MM standard. Values can be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- GEOGRAPHIC: This is for SRSs based on georeferenced coordinate systems with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axes of latitude, longitude (and elevation). These SRSs are of type PLANAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or ROUND EARTH.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PROJECTED: This is for SRSs based on georeferenced coordinate systems that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>do not have axes of latitude and longitude. These SRSs are of type PLANAR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ENGINEERING: This is for SRSs based on non-georeferenced coordinate systems</td>
</tr>
<tr>
<td></td>
<td></td>
<td>. These SRSs are of type PLANAR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- GEOCENTRIC: Unsupported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- COMPOUND: Unsupported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- VERTICAL: Unsupported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If srs_type is empty, the type is unspecified.</td>
</tr>
<tr>
<td>linear_unit_of_measure</td>
<td>UNSIGNED BIGINT</td>
<td>The linear unit of measure used by the spatial reference system.</td>
</tr>
<tr>
<td>angular_unit_of_measure</td>
<td>UNSIGNED BIGINT</td>
<td>The angular unit of measure used by the spatial reference system.</td>
</tr>
<tr>
<td>count_in_use</td>
<td>UNSIGNED BIGINT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>polygon_format</td>
<td>LONG VARCHAR</td>
<td>The orientation of the rings in a polygon. One of Counter-Clockwise, ClockW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is, or EvenOdd.</td>
</tr>
<tr>
<td>storage_format</td>
<td>LONG VARCHAR</td>
<td>Whether the data is stored in normalized format (Internal), unnormalized</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format (Original), or both (Mixed).</td>
</tr>
<tr>
<td>definition</td>
<td>LONG VARCHAR</td>
<td>The WKT definition of the spatial reference system in the format defined by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the OGC standard.</td>
</tr>
<tr>
<td>transform_definition</td>
<td>LONG VARCHAR</td>
<td>Transform definition settings for use when transforming data from this SRS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to another.</td>
</tr>
</tbody>
</table>
Constraints on underlying system table

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss_user_type</td>
<td>SMALLINT</td>
<td>The Adaptive Server Enterprise user type.</td>
</tr>
<tr>
<td>ss_domain_id</td>
<td>SMALLINT</td>
<td>The Adaptive Server Enterprise domain ID.</td>
</tr>
<tr>
<td>ss_type_name</td>
<td>VARCHAR (30)</td>
<td>The Adaptive Server Enterprise type name.</td>
</tr>
<tr>
<td>primary_sa_domain_id</td>
<td>SMALLINT</td>
<td>The corresponding SQL Anywhere primary domain ID.</td>
</tr>
<tr>
<td>primary_sa_user_type</td>
<td>SMALLINT</td>
<td>The corresponding SQL Anywhere primary user type.</td>
</tr>
</tbody>
</table>

See also

- “ST_SPATIAL_REFERENCE_SYSTEMS consolidated view” on page 1420
- “CREATE SPATIAL REFERENCE SYSTEM statement” on page 673

SYSSQLSERVERTYPE system view

The SYSSQLSERVERTYPE system view contains information relating to compatibility with Adaptive Server Enterprise. The underlying system table for this view is ISYSSQLSERVERTYPE.

SYSSUBSCRIPTION system view

Each row in the SYSSUBSCRIPTION system view describes a subscription from one user ID (which must have the REMOTE system privilege) to one publication. The underlying system table for this view is ISYSSUBSCRIPTION.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>UNSIGNED INT</td>
<td>The identifier for the publication to which the user ID is subscribed.</td>
</tr>
<tr>
<td>user_id</td>
<td>UNSIGNED INT</td>
<td>The ID of the user who is subscribed to the publication.</td>
</tr>
<tr>
<td>subscribe_by</td>
<td>CHAR(128)</td>
<td>The value of the SUBSCRIBE BY expression, if any, for the subscription.</td>
</tr>
<tr>
<td>created</td>
<td>UNSIGNED BIGINT</td>
<td>The offset in the transaction log at which the subscription was created.</td>
</tr>
<tr>
<td>started</td>
<td>UNSIGNED BIGINT</td>
<td>The offset in the transaction log at which the subscription was started.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

- PRIMARY KEY (publication_id, user_id, subscribe_by)
- FOREIGN KEY (publication_id) REFERENCES SYS.ISYSPUBLICATION (publication_id)
- FOREIGN KEY (user_id) REFERENCES SYS.ISYSUSER (user_id)

**SYSSYNC system view**

The SYSSYNC system view contains information relating to MobiLink synchronization. Some columns in this view contain potentially sensitive data. For that reason, access to this view is restricted. The SYSSYNC2 view provides public access to the data in this view except for the potentially sensitive columns. The underlying system table for this view is ISYSSYNC.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync_id</td>
<td>UNSIGNED INT</td>
<td>A number that uniquely identifies the row.</td>
</tr>
<tr>
<td>type</td>
<td>CHAR(1)</td>
<td>This value is always D.</td>
</tr>
<tr>
<td>publication_id</td>
<td>UNSIGNED INT</td>
<td>A publication_id found in the SYSPUBLICATION system view.</td>
</tr>
<tr>
<td>progress</td>
<td>UNSIGNED BIGINT</td>
<td>The log offset of the last successful upload.</td>
</tr>
<tr>
<td>site_name</td>
<td>CHAR(128)</td>
<td>A MobiLink user name.</td>
</tr>
<tr>
<td>&quot;option&quot;</td>
<td>LONG VARCHAR</td>
<td>Synchronization options.</td>
</tr>
<tr>
<td>server_connect</td>
<td>LONG VARCHAR</td>
<td>The address or URL of the MobiLink server.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>server_conn_type</td>
<td>LONG VARCHAR</td>
<td>The communication protocol, such as TCP/IP, to use when synchronizing.</td>
</tr>
<tr>
<td>last_download_time</td>
<td>TIMESTAMP</td>
<td>Indicates the last time a download stream was received from the MobiLink server.</td>
</tr>
<tr>
<td>last_upload_time</td>
<td>TIMESTAMP</td>
<td>Indicates the last time (measured at the MobiLink server) that information was successfully uploaded. The default is jan-1-1900.</td>
</tr>
<tr>
<td>created</td>
<td>UNSIGNED BIGINT</td>
<td>The log offset at which the subscription was created.</td>
</tr>
<tr>
<td>log_sent</td>
<td>UNSIGNED BIGINT</td>
<td>The log progress up to which information has been uploaded. It is not necessary that an acknowledgement of the upload be received for the entry in this column to be updated.</td>
</tr>
<tr>
<td>generation_number</td>
<td>INTEGER</td>
<td>For file-base downloads, the last generation number received for this subscription. The default is 0.</td>
</tr>
<tr>
<td>extended_state</td>
<td>VARCHAR(1024)</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>script_version</td>
<td>CHAR(128)</td>
<td>Indicates the script version used by the CREATE and ALTER SYNCHRONIZATION SUBSCRIPTION state-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ments and the START SYNCHRONIZATION SCHE-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MA CHANGE statement.</td>
</tr>
<tr>
<td>subscription_name</td>
<td>CHAR (128)</td>
<td>The name of the subscription.</td>
</tr>
<tr>
<td>server_protocol</td>
<td>UNSIGNED BIGINT</td>
<td>For internal use only. Contains a value used internally to identify the version of the MobiLink server.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

- PRIMARY KEY (sync_id)
- FOREIGN KEY (publication_id) REFERENCES SYS.ISYSPUBLICATION (publication_id)
- UNIQUE INDEX (publication_id, site_name)
- UNIQUE INDEX (subscription_name)

**SYSSYNCPROFILE system view**

The SYSSYNCPROFILE system view contains information relating to synchronization profiles for MobiLink synchronization.
The underlying system table for this view is ISYSSYNCPROFILE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the sync profile.</td>
</tr>
<tr>
<td>profile_name</td>
<td>CHAR(128)</td>
<td>The name of the sync profile.</td>
</tr>
<tr>
<td>profile_defn</td>
<td>LONG VARCHAR</td>
<td>The definition for the sync profile.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (object_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id)
- UNIQUE INDEX (profile_name)

**SYSSYNCSCRIPT system view**

Each row in the SYSSYNCSCRIPT system view identifies a stored procedure for MobiLink scripted upload. This view is almost identical to the SYSSYNCSCRIPTS view, except that the values in this view are in their raw format.

The underlying system table for this view is ISYSSYNCSCRIPT.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pub_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the publication to which the script belongs.</td>
</tr>
<tr>
<td>table_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the table to which the script applies.</td>
</tr>
<tr>
<td>type</td>
<td>UNSIGNED INT</td>
<td>The type of upload procedure.</td>
</tr>
<tr>
<td>proc_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the stored procedure to use for the publication.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (pub_object_id, table_object_id, type)
- FOREIGN KEY (pub_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
- FOREIGN KEY (table_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
- FOREIGN KEY (proc_object_id) REFERENCES SYS.ISYSOBJECT (object_id)
See also

- “Scripted upload” [MobiLink - Client Administration]
- “SYSSYNCSOURCES consolidated view” on page 1435
- “SYSPROCEDURE system view” on page 1377
- “SYSPUBLICATION system view” on page 1381

SYSTAB system view

Each row of the SYSTAB system view describes one table or view in the database. Additional information for views can be found in the SYSVIEW system view. The underlying system table for this view is ISYSTAB.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>Each table is assigned a unique number (the table number).</td>
</tr>
<tr>
<td>dbspace_id</td>
<td>SMALLINT</td>
<td>A value indicating which dbspace contains the table.</td>
</tr>
<tr>
<td>count</td>
<td>UNSIGNED BIGINT</td>
<td>The number of rows in the table or materialized view. This value is updated during each successful checkpoint. This number is used to optimize database access. The count is always 0 for a non-materialized view or remote table.</td>
</tr>
<tr>
<td>creator</td>
<td>UNSIGNED INT</td>
<td>The user number of the owner of the table or view.</td>
</tr>
<tr>
<td>table_page_count</td>
<td>INTEGER</td>
<td>The total number of main pages used by the underlying table.</td>
</tr>
<tr>
<td>ext_page_count</td>
<td>INTEGER</td>
<td>The total number of extension pages used by the underlying table.</td>
</tr>
<tr>
<td>commit_action</td>
<td>INTEGER</td>
<td>For global temporary tables, 0 indicates that the ON COMMIT PRESERVE ROWS clause was specified when the table was created, 1 indicates that the ON COMMIT DELETE ROWS clause was specified when the table was created (the default behavior for temporary tables), and 3 indicates that the NOT TRANSACTIONAL clause was specified when the table was created. For non-temporary tables, commit_action is always 0.</td>
</tr>
<tr>
<td>share_type</td>
<td>INTEGER</td>
<td>For global temporary tables, 4 indicates that the SHARE BY ALL clause was specified when the table was created, and 5 indicates that the SHARE BY ALL clause was not specified when the table was created. For non-temporary tables, share_type is always 5 because the SHARE BY ALL clause cannot be specified when creating non-temporary tables.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the table.</td>
</tr>
<tr>
<td>last_modified_at</td>
<td>TIMESTAMP</td>
<td>The local time at which the data in the table was last modified. This column is only updated at checkpoint time.</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>The name of the table or view. One creator cannot have two tables or views with the same name.</td>
</tr>
<tr>
<td>table_type</td>
<td>TINYINT</td>
<td>The type of table or view. Values include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1  Base table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 2  Materialized view</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 3  Global temporary table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 4  Local temporary table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 5  Text index base table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 6  Text index global temporary table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 21 View</td>
</tr>
<tr>
<td>replicate</td>
<td>CHAR(1)</td>
<td>This value is for internal use only.</td>
</tr>
<tr>
<td>server_type</td>
<td>TINYINT</td>
<td>The location of the data for the underlying table. Values include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1  Local server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 3  Remote server</td>
</tr>
<tr>
<td>tab_page_list</td>
<td>LONG VAR-BIT</td>
<td>For internal use only. The set of pages that contain information for the table, expressed as a bitmap.</td>
</tr>
<tr>
<td>ext_page_list</td>
<td>LONG VAR-BIT</td>
<td>For internal use only. The set of pages that contain row extensions and large object (LOB) pages for the table, expressed as a bitmap.</td>
</tr>
<tr>
<td>pct_free</td>
<td>UNSIGNED INT</td>
<td>The PCT_FREE specification for the table, if one has been specified; otherwise, NULL.</td>
</tr>
<tr>
<td>clustered_index_id</td>
<td>UNSIGNED INT</td>
<td>The ID of the clustered index for the table. If none of the indexes are clustered, then this field is NULL.</td>
</tr>
<tr>
<td>encrypted</td>
<td>CHAR(1)</td>
<td>Whether the table or materialized view is encrypted.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>last_modified_tsn</td>
<td>UNSIGNED BIGINT</td>
<td>A sequence number assigned to the transaction that modified the table. This column is only updated at checkpoint time.</td>
</tr>
<tr>
<td>current_schema</td>
<td>UNSIGNED INT</td>
<td>The current schema version of the table.</td>
</tr>
<tr>
<td>file_id</td>
<td>SMALLINT</td>
<td>DEPRECATED. This column is present in SYSVIEW, but not in the underlying system table ISYSTAB. The contents of this column is the same as dbspace_id and is provided for compatibility. Use dbspace_id instead.</td>
</tr>
<tr>
<td>table_type_str</td>
<td>CHAR(13)</td>
<td>Readable value for table_type. Values include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BASE Base table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MAT VIEW Materialized view</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GBL TEMP Global temporary table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• VIEW View</td>
</tr>
<tr>
<td>last_modified_at_utc</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>The UTC time at which the data in the table was last modified. This column is only updated at checkpoint time.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- FOREIGN KEY (dbspace_id) REFERENCES SYS.ISYSDBSPACE (dbspace_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id)
- PRIMARY KEY (table_id)
- FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
- UNIQUE INDEX (table_name, creator)

See also

- “SYSVIEW system view” on page 1416

SYSTABCOL system view

The SYSTABCOL system view contains one row for each column of each table and view in the database. The underlying system table for this view is ISYSTABCOL.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>The object ID of the table or view to which the column belongs.</td>
</tr>
<tr>
<td>column_id</td>
<td>UNSIGNED INT</td>
<td>The ID of the column. For each table, column numbering starts at 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The column_id value determines the order of columns in the result set when SELECT * is used. It also determines the column order for an INSERT statement when a list of column names is not provided.</td>
</tr>
<tr>
<td>domain_id</td>
<td>SMALLINT</td>
<td>The data type for the column, indicated by a data type number listed in the SYSDOMAIN system view.</td>
</tr>
<tr>
<td>nulls</td>
<td>CHAR(1)</td>
<td>Indicates whether NULL values are allowed in the column.</td>
</tr>
<tr>
<td>width</td>
<td>BIGINT</td>
<td>The length of a string column, the precision of numeric columns, or the number of bytes of storage for any other data type.</td>
</tr>
<tr>
<td>scale</td>
<td>SMALLINT</td>
<td>The number of digits after the decimal point for NUMERIC or DECIMAL data type columns. For string columns, a value of 1 indicates character-length semantics and 0 indicates byte-length semantics.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the table column.</td>
</tr>
<tr>
<td>max_identity</td>
<td>BIGINT</td>
<td>The largest value of the column, if it is an AUTOINCREMENT, IDENTITY, or GLOBAL AUTOINCREMENT column.</td>
</tr>
<tr>
<td>column_name</td>
<td>CHAR(128)</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>&quot;default&quot;</td>
<td>LONG VARCHAR</td>
<td>The default value for the column. This value, if specified, is only used when an INSERT statement does not specify a value for the column.</td>
</tr>
<tr>
<td>user_type</td>
<td>SMALLINT</td>
<td>The data type, if the column is defined using a user-defined data type.</td>
</tr>
<tr>
<td>column_type</td>
<td>CHAR(1)</td>
<td>The type of column (C=computed column, and R=other columns).</td>
</tr>
<tr>
<td>compressed</td>
<td>TINYINT</td>
<td>Whether this column is stored in a compressed format.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>collect_stats</td>
<td>TINYINT</td>
<td>Whether the system automatically collects and updates statistics on this column.</td>
</tr>
<tr>
<td>inline_max</td>
<td>SMALLINT</td>
<td>The maximum number of bytes of a BLOB to store in a row. A NULL value indicates that either the default value has been applied, or that the column is not a character or binary type. A non-NULL value corresponds to the INLINE value specified for the column using the CREATE TABLE or ALTER TABLE statement.</td>
</tr>
<tr>
<td>inline_long</td>
<td>SMALLINT</td>
<td>The number of duplicate bytes of a BLOB to store in a row if the BLOB size exceeds the inline_max value. A NULL value indicates that either the default value has been applied, or that the column is not a character or binary type. A non-NULL value corresponds to the PREFIX value specified for the column using the CREATE TABLE or ALTER TABLE statement.</td>
</tr>
<tr>
<td>lob_index</td>
<td>TINYINT</td>
<td>Whether to build indexes on BLOB values in the column that exceed an internal threshold size (approximately eight database pages). A NULL value indicates either that the default is applied, or that the column is not BLOB type. A value of 1 indicates that indexes will be built. A value of 0 indicates that no indexes will be built. A non-NULL value corresponds to whether INDEX or NO INDEX was specified for the column using the CREATE TABLE or ALTER TABLE statement.</td>
</tr>
<tr>
<td>base_type_str</td>
<td>VARCHAR(32,767)</td>
<td>The annotated type string representing the physical type of the column.</td>
</tr>
<tr>
<td>nonmaterialized_value</td>
<td>LONG BINARY</td>
<td>Internal use only.</td>
</tr>
<tr>
<td>start_schema</td>
<td>UNSIGNED INT</td>
<td>The first version of the table schema in which this column exists.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (table_id, column_id)
- FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)
- FOREIGN KEY (domain_id) REFERENCES SYS.ISYSDOMAIN (domain_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
- FOREIGN KEY (user_type) REFERENCES SYS.ISYSUSERTYPE (type_id)
SYSTABLEPERM system view

Privileges granted on tables and views by the GRANT statement are stored in the SYSTABLEPERM system view. Each row in this view corresponds to one table, one user ID granting the privilege (grantor) and one user ID granted the privilege (grantee). The underlying system table for this view is ISYSTABLEPERM.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stable_id</td>
<td>UNSIGNED INT</td>
<td>The table number of the table or view to which the privileges apply.</td>
</tr>
<tr>
<td>grantee</td>
<td>UNSIGNED INT</td>
<td>The user number of the user ID receiving the privilege.</td>
</tr>
<tr>
<td>grantor</td>
<td>UNSIGNED INT</td>
<td>The user number of the user ID granting the privilege.</td>
</tr>
<tr>
<td>selectauth</td>
<td>CHAR(1)</td>
<td>Indicates whether SELECT privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.</td>
</tr>
<tr>
<td>insertauth</td>
<td>CHAR(1)</td>
<td>Indicates whether INSERT privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.</td>
</tr>
<tr>
<td>deleteauth</td>
<td>CHAR(1)</td>
<td>Indicates whether DELETE privileges has been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.</td>
</tr>
<tr>
<td>updateauth</td>
<td>CHAR(1)</td>
<td>Indicates whether UPDATE privileges have been granted for all columns in the table. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.</td>
</tr>
<tr>
<td>updatecols</td>
<td>CHAR(1)</td>
<td>Indicates whether UPDATE privileges have only been granted for some of the columns in the underlying table. If updatecols has the value Y, there will be one or more rows in the SYSCOLPERM system view granting update privileges for the columns.</td>
</tr>
<tr>
<td>alterauth</td>
<td>CHAR(1)</td>
<td>Indicates whether ALTER privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>referenc-</td>
<td>CHAR(1)</td>
<td>Indicates whether REFERENCE privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.</td>
</tr>
<tr>
<td>auth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loadauth</td>
<td>CHAR(1)</td>
<td>Indicates whether LOAD privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.</td>
</tr>
<tr>
<td>truncateauth</td>
<td>CHAR(1)</td>
<td>Indicates whether TRUNCATE privileges have been granted. Possible values are Y, N, or G. See the Remarks area below for more information about what these values mean.</td>
</tr>
</tbody>
</table>

**Remarks**

There are several types of privileges that can be granted. Each privilege can have one of the following three values.

- **N**  No, the grantee has not been granted this privilege by the grantor.
- **Y**  Yes, the grantee has been given this privilege by the grantor.
- **G**  The grantee has been given this privilege and can grant the same privilege to another user.

**Note**

The grantee might have been given the privilege for the same table by another grantor. If so, this information would be found in a different row of the SYSTABLEPERM system view.

**Constraints on underlying system table**

| PRIMARY KEY (stable_id, grantee, grantor) |
| FOREIGN KEY (stable_id) REFERENCES SYS.ISYSTAB (table_id) |
| FOREIGN KEY (grantor) REFERENCES SYS.ISYSUSER (user_id) |
| FOREIGN KEY (grantee) REFERENCES SYS.ISYSUSER (user_id) |

**See also**

- “GRANT statement” on page 827

**SYSTEXTCONFIG system view**

Each row in the SYSTEXTCONFIG system view describes one text configuration object, for use with the full text search feature. The underlying system table for this view is ISYSTEXTCONFIG.
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID for the text configuration object.</td>
</tr>
<tr>
<td>creator</td>
<td>UNSIGNED INT</td>
<td>The creator of the text configuration object.</td>
</tr>
<tr>
<td>term_breaker</td>
<td>TINYINT</td>
<td>The algorithm used to separate a string into terms or words. Values are 0 for GENERIC and 1 for NGRAM. With GENERIC, any string of one or more alphanumeric characters separated by non-alphanumerics are treated as a term. NGRAM is for approximate matching or for documents that do not use a whitespace to separate terms.</td>
</tr>
<tr>
<td>stemmer</td>
<td>TINYINT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>min_term_length</td>
<td>TINYINT</td>
<td>The minimum length, in characters, allowed for a term. Terms that are shorter than min_term_length are ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The MINIMUM TERM LENGTH setting is only meaningful for the GENERIC term breaker. For NGRAM text indexes, the setting is ignored.</td>
</tr>
<tr>
<td>max_term_length</td>
<td>TINYINT</td>
<td>For GENERIC text indexes, the maximum length, in characters, allowed for a term. Terms that are longer than max_term_length are ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For NGRAM text indexes, this is the length of the n-grams into which terms are broken.</td>
</tr>
<tr>
<td>collation</td>
<td>CHAR(128)</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>text_config_name</td>
<td>CHAR(128)</td>
<td>The name of the text configuration object.</td>
</tr>
<tr>
<td>prefilter</td>
<td>LONG VARCHAR</td>
<td>The function and library name for an external prefilter library.</td>
</tr>
<tr>
<td>postfilter</td>
<td>LONG VARCHAR</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>char_stoplist</td>
<td>LONG VARCHAR</td>
<td>Terms to ignore when performing a full text search on CHAR columns. These terms are also omitted from text indexes. This column is used when the text configuration object is created from default_char.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>nchar_stoplist</td>
<td>LONG NVARCHAR</td>
<td>Terms to ignore when performing a full text search on NCHAR columns. These terms are also omitted from text indexes. This column is used when the text configuration object is created from default_nchar.</td>
</tr>
<tr>
<td>external_term_breaker</td>
<td>LONG VARCHAR</td>
<td>The function and library name for an external term breaker library.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

- PRIMARY KEY (object_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
- FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
- UNIQUE INDEX (creator, text_config_name)

**See also**

- “What to specify when creating or altering text configuration objects” [SQL Anywhere Server - SQL Usage]
- “Full text search” [SQL Anywhere Server - SQL Usage]

**SYSTEXTIDX system view**

Each row in the SYSTEXTIDX system view describes one text index. The underlying system table for this view is ISYSTEXTIDX.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the text index in SYSIDX.</td>
</tr>
<tr>
<td>sequence</td>
<td>UNSIGNED INT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>status</td>
<td>UNSIGNED INT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>text_config</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the text configuration object in SYSTEXTCONFIG.</td>
</tr>
<tr>
<td>next_handle</td>
<td>UNSIGNED INT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>last_handle</td>
<td>UNSIGNED INT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>deleted_length</td>
<td>UNSIGNED BIGINT</td>
<td>The total size of deleted indexed values in the text index.</td>
</tr>
</tbody>
</table>
### Column name | Data type | Description
--- | --- | ---
pending_length | UNSIGNED BIGINT | The total size of indexed values that will be added to the text index at the next refresh.
refresh_type | TINYINT | The type of refresh. One of:
- 1 MANUAL
- 2 AUTO
- 3 IMMEDIATE
refresh_interval | UNSIGNED INT | The AUTO REFRESH interval, in minutes.
last_refresh | TIMESTAMP | The local time of the last refresh.
last_refresh_utc | TIMESTAMP WITH TIME ZONE | The UTC time of the last refresh.

**Constraints on underlying system table**

PRIMARY KEY (index_id, sequence)

FOREIGN KEY (index_id) REFERENCES SYS.ISYSOBJECT (object_id)

FOREIGN KEY (text_config) REFERENCES SYS.ISYSTEXTCONFIG (object_id)

**See also**

- “Full text search” [SQL Anywhere Server - SQL Usage]

### SYSTEXTIDXTAB system view

Each row in the SYSTEXTIDXTAB system view describes a generated table that is part of a text index. The underlying system table for this view is ISYSTEXTIDXTAB.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index_id</td>
<td>UNSIGNED BIGINT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>sequence</td>
<td>UNSIGNED INT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>table_type</td>
<td>UNSIGNED INT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>For internal use only.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

PRIMARY KEY (index_id, sequence, table_type)
FOREIGN KEY (index_id, sequence) REFERENCES SYS.ISYSTEXTIDX (index_id, sequence)

FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)

See also

- “Full text search” [SQL Anywhere Server - SQL Usage]

**SYSTRIGGER system view**

Each row in the SYSTRIGGER system view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE). The underlying system table for this view is ISYSTRIGGER.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger_id</td>
<td>UNSIGNED INT</td>
<td>A unique number for the trigger in the SYSTRIGGER view.</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>The table ID of the table to which this trigger belongs.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID for the trigger in the database.</td>
</tr>
<tr>
<td>event</td>
<td>CHAR(1)</td>
<td>The operation that causes the trigger to fire.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● A INSERT, DELETE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● B INSERT, UPDATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● C UPDATE COLUMNS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● D DELETE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● E DELETE, UPDATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● I INSERT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● M INSERT, DELETE, UPDATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● U UPDATE</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>trigger_time</td>
<td>CHAR(1)</td>
<td>The time when the trigger fires relative to the event.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>A</strong>  AFTER (row-level trigger)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>B</strong>  BEFORE (row-level trigger)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>I</strong>  INSTEAD OF (row-level trigger)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>K</strong>  INSTEAD OF (statement-level trigger)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>R</strong>  RESOLVE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>S</strong>  AFTER (statement-level trigger)</td>
</tr>
<tr>
<td>trigger_order</td>
<td>SMALLINT</td>
<td>The order in which are fired when there are multiple triggers of the same type (insert, update, or delete) set to fire at the same time (applies to BEFORE or AFTER triggers only, only).</td>
</tr>
<tr>
<td>foreign_table_id</td>
<td>UNSIGNED INT</td>
<td>The ID of the table containing a foreign key definition that has a referential triggered action (such as ON DELETE CASCADE). The foreign_table_id value reflects the value of ISYSIDX.table_id.</td>
</tr>
<tr>
<td>foreign_key_id</td>
<td>UNSIGNED INT</td>
<td>The ID of the foreign key for the table referenced by foreign_table_id. The foreign_key_id value reflects the value of ISYSIDX.index_id.</td>
</tr>
<tr>
<td>referential_action</td>
<td>CHAR(1)</td>
<td>The action defined by a foreign key. This single-character value corresponds to the action that was specified when the foreign key was created.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>C</strong>  CASCADE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>D</strong>  SET DEFAULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>N</strong>  SET NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>R</strong>  RESTRICT</td>
</tr>
<tr>
<td>trigger_name</td>
<td>CHAR(128)</td>
<td>The name of the trigger. One table cannot have two triggers with the same name.</td>
</tr>
<tr>
<td>trigger_defn</td>
<td>LONG VARCHAR</td>
<td>The command that was used to create the trigger.</td>
</tr>
<tr>
<td>remarks</td>
<td>LONG VARCHAR</td>
<td>Remarks about the trigger. This value is stored in the ISYSREMARK system table.</td>
</tr>
</tbody>
</table>
### Column name | Data type | Description
--- | --- | ---
source | LONG VARCHAR | The SQL source for the trigger. This value is stored in the ISYS-SOURCE system table.

#### Constraints on underlying system table
- PRIMARY KEY (trigger_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
- FOREIGN KEY (table_id) REFERENCES SYS.ISYSTAB (table_id)
- FOREIGN KEY fkey_index (foreign_table_id, foreign_key_id) REFERENCES SYS.ISYSIDX (table_id, index_id)
- UNIQUE INDEX (table_id, event, trigger_time, trigger_order)
- UNIQUE INDEX (trigger_name, table_id)
- UNIQUE INDEX (table_id, foreign_table_id, foreign_key_id, event)

### SYSTYPEMAP system view
The SYSTYPEMAP system view contains the compatibility mapping values for entries in the SYSSQLSERVERTYPE system view. The underlying system table for this view is ISYSTYPEMAP.

| Column name | Data type | Description |
--- | --- | ---
ss_user_type | SMALLINT | Contains the Adaptive Server Enterprise user type. |
sa_domain_id | SMALLINT | Contains the corresponding SQL Anywhere domain_id. |
sa_user_type | SMALLINT | Contains the corresponding SQL Anywhere user type. |
nullable | CHAR(1) | Whether the type allows NULL values. |

#### Constraints on underlying system table
- FOREIGN KEY (sa_domain_id) REFERENCES SYS.ISYSDOMAIN (domain_id)

### SYSUNITOFMEASURE system view
Each row of the SYSUNITOFMEASURE system view describes a unit of measure defined in the database. The underlying table for the SYSUNITOFMEASURE system view is the ISYSUNITOFMEASURE system table.
### Column name | Data type | Description
---|---|---
object_id | UNSIGNED BIGINT | For system use only.
owner | UNSIGNED INT | The owner of the unit of measure.
unit_name | CHAR(128) | The name of the unit of measure.
unit_type | CHAR(7) | Angular or linear.
conversion_factor | DOUBLE | The conversion factor for the unit of measure.

**Constraints on underlying system table**

- PRIMARY KEY (object_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id)
- FOREIGN KEY (owner) REFERENCES SYS.ISYSUSER (user_id)
- UNIQUE CONSTRAINT (unit_name)

### SYSUSER system view

Each row in the SYSUSER system view describes a user in the system. Standalone roles are also stored in this view as well, but only the user_id, object_id, user_name, and user_type columns are meaningful for these roles. The underlying system table for this view is ISYSUSER.

| Column name | Data type | Description |
---|---|---|
user_id | UNSIGNED INT | A unique identifier for the user assigned to the login policy. |
object_id | UNSIGNED BIGINT | A unique identifier for the user in the database. |
user_name | CHAR(128) | The login name for the user. |
pASSWORD | BINARY(128) | The password for the user. For security, data in this column is visible only to users with the SELECT ANY TABLE system privilege. |
login_policy_id | UNSIGNED BIGINT | A unique identifier for the login policy. |
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expired_password_on_login</td>
<td>TINYINT</td>
<td>A value that indicates if the password for the user expires at the next login.</td>
</tr>
<tr>
<td>password_creation_time</td>
<td>TIMESTAMP</td>
<td>The local time that the password was created for the user.</td>
</tr>
<tr>
<td>failed_login_attempts</td>
<td>UNSIGNED INT</td>
<td>The number of times that a user can fail to log in before the account is locked.</td>
</tr>
<tr>
<td>last_login_time</td>
<td>TIMESTAMP</td>
<td>The local time that the user last logged in.</td>
</tr>
</tbody>
</table>
| user_type                   | TINYINT          | A value that indicates whether the user is a regular user, or a role, or a user extended as a role. And whether the user, role, or extended role can be altered (mutable) or removed. Possible values:  
  • 1  Immutable system role.  
  • 5  Mutable system role  
  • 9  Immutable and removable system role.  
  • 12  Mutable and removable user.  
  • 13  Mutable and removable role.  
  • 14  Mutable and removable user extended as role. |
<p>| user_dn                     | CHAR (1024)      | An LDAP Distinguished Name (DN) identifier for the user that is unique within a domain and across domains. The DN is used to authenticate with an LDAP server. |</p>
<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_dn_cached_at</td>
<td>TIMESTAMP</td>
<td>The time that the user_dn column was last cached. This value is used to determine whether to purge an old DN. Regardless of the database server local time zone, the value is stored in Coordinated Universal Time (UTC).</td>
</tr>
<tr>
<td>password_creation_time_utc</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>The UTC time that the password was created for the user.</td>
</tr>
<tr>
<td>last_login_time_utc</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>The UTC time that the user last logged in.</td>
</tr>
<tr>
<td>dual_password</td>
<td>BINARY(128)</td>
<td>The first and/or second parts of the dual password for the user. For security, data in this column is visible only to users with the SELECT ANY TABLE system privilege.</td>
</tr>
</tbody>
</table>

**Note**
For databases created using SQL Anywhere 16 or later, the underlying system table for this view is always encrypted to protect the data from unauthorized access.

**Constraints on underlying system table**

- PRIMARY KEY (user_id)

- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

- FOREIGN KEY (login_policy_id) REFERENCES SYS.ISYSLOGINPOLICY (login_policy_id)

- UNIQUE INDEX (user_name)

**See also**
- “sp_sys_priv_role_info system procedure” on page 1311
- “SYSLOGINPOLICY system view” on page 1371
- “SYSLOGINPOLICYOPTION system view” on page 1371

**SYSUSERMESSAGE system view**

Each row in the SYSUSERMESSAGE system view holds a user-defined message for an error condition. The underlying system table for this view is ISYSUSERMESSAGE.
Previous versions of the catalog contained a SYSUSERMESSAGES system table. That table has been renamed to be ISYSUSERMESSAGE (without an 'S'), and is the underlying table for this view.

### Note

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>INTEGER</td>
<td>A unique identifying number for the error condition.</td>
</tr>
<tr>
<td>uid</td>
<td>UNSIGNED INT</td>
<td>The user number that defined the message.</td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR(255)</td>
<td>The message corresponding to the error condition.</td>
</tr>
<tr>
<td>langid</td>
<td>SMALLINT</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**Constraints on underlying system table**

FOREIGN KEY (uid) REFERENCES SYS.ISYSUSER (user_id)

UNIQUE CONSTRAINT (error, langid)

### SYSUSERTYPE system view

Each row in the SYSUSERTYPE system view holds a description of a user-defined data type. The underlying system table for this view is ISYSUSERTYPE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_id</td>
<td>SMALLINT</td>
<td>A unique identifying number for the user-defined data type.</td>
</tr>
<tr>
<td>creator</td>
<td>UNSIGNED INT</td>
<td>The user number of the owner of the data type.</td>
</tr>
<tr>
<td>domain_id</td>
<td>SMALLINT</td>
<td>The data type on which this user defined data type is based, indicated by a data type number listed in the SYSDOMAIN system view.</td>
</tr>
<tr>
<td>nulls</td>
<td>CHAR(1)</td>
<td>Whether the user-defined data type allows nulls. Possible values are Y, N, or U. A value of U indicates that nullability is unspecified.</td>
</tr>
<tr>
<td>width</td>
<td>BIGINT</td>
<td>The length of a string column, the precision of a numeric column, or the number of bytes of storage for any other data type.</td>
</tr>
<tr>
<td>scale</td>
<td>SMALLINT</td>
<td>The number of digits after the decimal point for numeric data type columns, and zero for all other data types.</td>
</tr>
<tr>
<td>type_name</td>
<td>CHAR(128)</td>
<td>The name for the data type.</td>
</tr>
<tr>
<td>&quot;default&quot;</td>
<td>LONG VARCHAR</td>
<td>The default value for the data type.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;check&quot;</td>
<td>LONG VARCHAR</td>
<td>The CHECK condition for the data type.</td>
</tr>
<tr>
<td>base_type_str</td>
<td>VARCHAR(32767)</td>
<td>The annotated type string representing the physical type of the user type.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (type_id)
- FOREIGN KEY (creator) REFERENCES SYS.ISYSUSER (user_id)
- FOREIGN KEY (domain_id) REFERENCES SYS.ISYSDOMAIN (domain_id)
- UNIQUE CONSTRAINT (type_name)

**SYSVIEW system view**

Each row in the SYSVIEW system view describes a view in the database. Additional information about views can also be found in the SYSTAB system view. The underlying system table for this view is ISYSVIEW.

You can also use the sa_materialized_view_info system procedure for a more readable format of the information for materialized views.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>view_object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The object ID of the view.</td>
</tr>
<tr>
<td>view_def</td>
<td>LONG VARCHAR</td>
<td>The definition (query specification) of the view.</td>
</tr>
<tr>
<td>mv_build_type</td>
<td>TINYINT</td>
<td>Currently unused.</td>
</tr>
<tr>
<td>mv_refresh_type</td>
<td>TINYINT</td>
<td>The refresh type defined for the view. Possible values are IMMEDIATE (1) and MANUAL (2).</td>
</tr>
<tr>
<td>mv_use_in_optimization</td>
<td>TINYINT</td>
<td>Whether the materialized view can be used during query optimization (0=cannot be used in optimization, 1=can be used in optimization)</td>
</tr>
<tr>
<td>mv_last.refreshed_at</td>
<td>TIMESTAMP</td>
<td>Indicates the local date and time that the materialized view was last refreshed.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mv_known_stale_at</td>
<td>TIME-STAMP</td>
<td>The local time at which the materialized view became stale. This value corresponds to the time at which one of the underlying base tables was detected as having changed. A value of 0 indicates that the view is either fresh, or that it has become stale but the database server has not marked it as such because the view has not been used since it became stale. Use the sa_materialized_view_info system procedure to determine the status of a materialized view.</td>
</tr>
<tr>
<td>mv_last_refreshed_tsn</td>
<td>UNSIGNED BIGINT</td>
<td>The sequence number assigned to the transaction that refreshed the materialized view.</td>
</tr>
<tr>
<td>mv_last_refreshed_at_utc</td>
<td>TIME-STAMP WITH TIME ZONE</td>
<td>Indicates the UTC date and time that the materialized view was last refreshed.</td>
</tr>
<tr>
<td>mv_known_stale_at_utc</td>
<td>TIME-STAMP WITH TIME ZONE</td>
<td>The UTC time at which the materialized view became stale. This value corresponds to the time at which one of the underlying base tables was detected as having changed. A value of 0 indicates that the view is either fresh, or that it has become stale but the database server has not marked it as such because the view has not been used since it became stale. Use the sa_materialized_view_info system procedure to determine the status of a materialized view. This column contains 0 when mv_last_refreshed_at is 0 and NULL when mv_last刷新ed_at is NULL.</td>
</tr>
</tbody>
</table>

**Remarks**

When a materialized view is refreshed with SNAPSHOT isolation, mv_last_refreshed_at and mv_last_refreshed_tsn refer to the earliest transaction that modified any row used during the computation of the materialized view contents.

**Constraints on underlying system table**

```
PRIMARY KEY (view_object_id)

FOREIGN KEY (view_object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
```
SYSWEBSERVICE system view

Each row in the SYSWEBSERVICE system view holds a description of a web service. The underlying system table for this view is ISYSWEBSERVICE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifying number for the web service.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>The ID of the web service.</td>
</tr>
<tr>
<td>service_name</td>
<td>CHAR(128)</td>
<td>The name assigned to the web service.</td>
</tr>
<tr>
<td>service_type</td>
<td>VARCHAR(40)</td>
<td>The type of the service; for example, RAW, HTTP, XML, SOAP, or DISH.</td>
</tr>
<tr>
<td>auth_required</td>
<td>CHAR(1)</td>
<td>Whether all requests must contain a valid user name and password.</td>
</tr>
<tr>
<td>secure_required</td>
<td>CHAR(1)</td>
<td>Whether insecure connections, such as HTTP, are to be accepted, or only secure connections, such as HTTPS.</td>
</tr>
<tr>
<td>url_path</td>
<td>CHAR(1)</td>
<td>Controls the interpretation of URLs.</td>
</tr>
<tr>
<td>user_id</td>
<td>UNSIGNED INT</td>
<td>If authentication is enabled, identifies the user, or group of users, that have permission to use the service. If authentication is disabled, specifies the account to use when processing requests.</td>
</tr>
<tr>
<td>parameter</td>
<td>LONG VARCHAR</td>
<td>A prefix that identifies the SOAP services to be included in a DISH service.</td>
</tr>
<tr>
<td>statement</td>
<td>LONG VARCHAR</td>
<td>A SQL statement that is always executed in response to a request. If NULL, arbitrary statements contained in each request are executed instead. Ignored for services of type DISH.</td>
</tr>
<tr>
<td>remarks</td>
<td>LONG VARCHAR</td>
<td>Remarks about the webservice. This value is stored in the ISYSREMARK system table.</td>
</tr>
</tbody>
</table>
### Column name | Data type | Description
---|---|---
enabled | CHAR(1) | Indicates whether the web service is currently enabled or disabled (see CREATE SERVICE).

#### Constraints on underlying system table
- PRIMARY KEY (service_id)
- FOREIGN KEY (object_id) REFERENCES SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL
- UNIQUE CONSTRAINT (service_name)

### Consolidated views
Consolidated views provide data in a form more frequently required by users. For example, consolidated views often provide commonly-needed joins. Consolidated views differ from system views in that they are not just a straightforward view of raw data in a underlying system table(s). For example, many of the columns in the system views are unintelligible ID values, whereas in the consolidated views, they are readable names.

### ST_GEOMETRY_COLUMNS consolidated view

This is a test... Each row of the ST_GEOMETRY_COLUMNS system view describes a spatial column defined in the database.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_catalog</td>
<td>VARCHAR(128)</td>
<td>For internal use.</td>
</tr>
<tr>
<td>table_schema</td>
<td>CHAR(128)</td>
<td>The name of the schema to which the table containing the spatial column belongs. This is equivalent to the table owner.</td>
</tr>
<tr>
<td>table_name</td>
<td>CHAR(128)</td>
<td>The name of the table containing the spatial column.</td>
</tr>
<tr>
<td>column_name</td>
<td>CHAR(128)</td>
<td>The name of the spatial column.</td>
</tr>
<tr>
<td>srs_name</td>
<td>CHAR(128)</td>
<td>The name of the SRS that is associated with the spatial column. If an SRS is not associated with the column, then srs_name is NULL.</td>
</tr>
<tr>
<td>srs_id</td>
<td>INTEGER</td>
<td>The SRID for the SRS associated with the spatial column.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>The numeric identifier for the table containing the column.</td>
</tr>
<tr>
<td>column_id</td>
<td>UNSIGNED INT</td>
<td>The numeric identifier for the column.</td>
</tr>
<tr>
<td>geometry_type_name</td>
<td>VARCHAR(32767)</td>
<td>The spatial data type of the geometries contained in the column (for example, ST_Point, ST_Geometry, and so on).</td>
</tr>
</tbody>
</table>

See also

- “Spatial reference systems (SRS) and Spatial reference identifiers (SRID)” [SQL Anywhere Server - Spatial Data Support]
- “Supported spatial data types and their hierarchy” [SQL Anywhere Server - Spatial Data Support]

**ST_SPATIAL_REFERENCE_SYSTEMS consolidated view**

Each row of the ST_SPATIAL_REFERENCE_SYSTEMS system view describes an SRS defined in the database. This view offers a slightly different amount of information than the SYSSPATIALREFERENCESYSTEM system view.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>owner</td>
<td>UNSIGNED INT</td>
<td>The owner of the SRS.</td>
</tr>
<tr>
<td>srs_name</td>
<td>CHAR(128)</td>
<td>The name of the SRS.</td>
</tr>
<tr>
<td>srs_id</td>
<td>INTEGER</td>
<td>The numeric identifier (SRID) for the spatial reference system.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>srs_type</td>
<td>CHAR(11)</td>
<td>The type of SRS as defined by the SQL/MM standard. Values can be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>GEOGRAPHIC</strong>  This is for SRSs based on georeferenced coordinate systems with axes of latitude, longitude (and elevation). These SRSs are of type PLANAR or ROUND EARTH.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>PROJECTED</strong>  This is for SRSs based on georeferenced coordinate systems that do not have axes of latitude and longitude. These SRSs are of type PLANAR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>ENGINEERING</strong>  This is for SRSs based on non-georeferenced coordinate systems. These SRSs are of type PLANAR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>GEOCENTRIC</strong>  Unsupported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>COMPOUND</strong>  Unsupported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>VERTICAL</strong>  Unsupported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If srs_type is empty, the type is unspecified.</td>
</tr>
<tr>
<td>round_earth</td>
<td>CHAR(1)</td>
<td>Whether the SRS type is ROUND EARTH (Y) or PLANAR (N).</td>
</tr>
<tr>
<td>axis_order</td>
<td>CHAR(12)</td>
<td>Describes how the database server interprets points with regards to latitude and longitude (for example when using the ST_Lat and ST_Long methods). For non-geographic spatial reference systems, the axis order is x/y/z/m. For geographic spatial reference systems, the default axis order is long/lat/z/m; lat/long/z/m is also supported.</td>
</tr>
<tr>
<td>snap_to_grid</td>
<td>DOUBLE</td>
<td>Defines the size of the grid SQL Anywhere uses when performing calculations.</td>
</tr>
<tr>
<td>tolerance</td>
<td>DOUBLE</td>
<td>Defines the precision to use when comparing points.</td>
</tr>
<tr>
<td>semi_major_axis</td>
<td>DOUBLE</td>
<td>Distance from center of the ellipsoid to the equator for a ROUND EARTH SRS.</td>
</tr>
<tr>
<td>semi_minor_axis</td>
<td>DOUBLE</td>
<td>Distance from center of the ellipsoid to the poles for a ROUND EARTH SRS.</td>
</tr>
<tr>
<td>Column name</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inv_flattening</td>
<td>DOUBLE</td>
<td>The inverse flattening used for the ellipsoid in a ROUND EARTH SRS. This is a ratio created by the following equation: ( 1/f = (\text{semi-major-axis}) / (\text{semi-major-axis} - \text{semi-minor-axis}) )</td>
</tr>
<tr>
<td>min_x</td>
<td>DOUBLE</td>
<td>The minimum x value allowed in coordinates.</td>
</tr>
<tr>
<td>max_x</td>
<td>DOUBLE</td>
<td>The maximum x value allowed in coordinates.</td>
</tr>
<tr>
<td>min_y</td>
<td>DOUBLE</td>
<td>The minimum y value allowed in coordinates.</td>
</tr>
<tr>
<td>max_y</td>
<td>DOUBLE</td>
<td>The maximum y value allowed in coordinates.</td>
</tr>
<tr>
<td>min_z</td>
<td>DOUBLE</td>
<td>The minimum z value allowed in coordinates.</td>
</tr>
<tr>
<td>max_z</td>
<td>DOUBLE</td>
<td>The maximum z value allowed in coordinates.</td>
</tr>
<tr>
<td>min_m</td>
<td>DOUBLE</td>
<td>The minimum m value allowed in coordinates.</td>
</tr>
<tr>
<td>max_m</td>
<td>DOUBLE</td>
<td>The maximum m value allowed in coordinates.</td>
</tr>
<tr>
<td>min_lat</td>
<td>DOUBLE</td>
<td>The minimum latitude value allowed for coordinates.</td>
</tr>
<tr>
<td>max_lat</td>
<td>DOUBLE</td>
<td>The maximum latitude value allowed for coordinates.</td>
</tr>
<tr>
<td>min_long</td>
<td>DOUBLE</td>
<td>The minimum longitude value allowed in coordinates.</td>
</tr>
<tr>
<td>max_long</td>
<td>DOUBLE</td>
<td>The maximum longitude value allowed in coordinates.</td>
</tr>
<tr>
<td>organization</td>
<td>LONG VARCHAR</td>
<td>The name of the organization that created the coordinate system used by the spatial reference system.</td>
</tr>
<tr>
<td>organization_coord-</td>
<td>INTEGRER</td>
<td>The ID given to the coordinate system by the organization that created it.</td>
</tr>
<tr>
<td>sys_id</td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear_unit_of_measure</td>
<td>CHAR(128)</td>
<td>The linear unit of measurement used by the SRS.</td>
</tr>
<tr>
<td>angular_unit_of_measure</td>
<td>CHAR(128)</td>
<td>The angular unit of measurement used by the SRS.</td>
</tr>
<tr>
<td>polygon_format</td>
<td>LONG VARCHAR</td>
<td>The orientation of the rings in a polygon. One of Counter-Clockwise, ClockWise, or EvenOdd.</td>
</tr>
<tr>
<td>storage_format</td>
<td>LONG VARCHAR</td>
<td>Whether the data is stored in normalized format (Internal), unnormalized format (Original), or both (Mixed).</td>
</tr>
</tbody>
</table>
### Consolidated views

#### ST_UNITS_OF_MEASURE consolidated view

Each row of the ST_UNITS_OF_MEASURE system view describes a unit of measure defined in the database. This view offers more information than the SYSUNITOFMEASURE system view.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object_id</td>
<td>UNSIGNED BIGINT</td>
<td>For system use only.</td>
</tr>
<tr>
<td>owner</td>
<td>UNSIGNED INT</td>
<td>The owner of the unit of measure.</td>
</tr>
<tr>
<td>unit_name</td>
<td>CHAR(128)</td>
<td>The name of the unit of measure.</td>
</tr>
<tr>
<td>unit_type</td>
<td>CHAR(7)</td>
<td>Angular or linear.</td>
</tr>
<tr>
<td>conversion_factor</td>
<td>DOUBLE</td>
<td>The conversion factor for the unit of measure.</td>
</tr>
<tr>
<td>description</td>
<td>LONG VARCHAR</td>
<td>Description for the unit of measure.</td>
</tr>
</tbody>
</table>

#### SYSARTICLECOLS consolidated view

Each row in the SYSARTICLECOLS view identifies a column in an article.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSARTICLECOLS"
as select p.publication_name, t.table_name, c.column_name
from SYS.ISYSARTICLECOL as ac
  join SYS.ISYSPUBLICATION as p on p.publication_id = ac.publication_id
  join SYS.ISYSTAB as t on t.table_id = ac.table_id
  join SYS.ISYSTABCOL as c on c.table_id = ac.table_id
  and c.column_id = ac.column_id
```
See also

- “SYSARTICLECOL system view” on page 1348
- “SYSPUBLICATION system view” on page 1381
- “SYSTAB system view” on page 1399
- “SYSTABCOL system view” on page 1401

SYSARTICLES consolidated view

Each row in the SYSARTICLES view describes an article in a publication.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSARTICLES"
    as select u1.user_name as publication_owner,p.publication_name,
              u2.user_name as table_owner,t.table_name,
              a.where_expr,a.subscribe_by_expr,a.alias
    from SYS.ISYSARTICLE as a
    join SYS.ISYSPUBLICATION as p on(a.publication_id = p.publication_id)
    join SYS.ISYSTAB as t on(a.table_id = t.table_id)
    join SYS.ISYSUSER as u1 on(p.creator = u1.user_id)
    join SYS.ISYSUSER as u2 on(t.creator = u2.user_id)
```

See also

- “SYSARTICLE system view” on page 1348
- “SYSPUBLICATION system view” on page 1381
- “SYSTAB system view” on page 1399
- “SYSUSER system view” on page 1412

SYSCAPABILITIES consolidated view

Each row in the SYSCAPABILITIES view specifies the status of a capability for a remote database server. This view gets its data from the ISYSCAPABILITY and ISYSCAPABILITYNAME system tables.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSCAPABILITIES"
    as select ISYSCAPABILITY.capid,ISYSCAPABILITY.srvid,
              property('RemoteCapability',ISYSCAPABILITY.capid) as capname,
              ISYSCAPABILITY.capvalue
    from SYS.ISYSCAPABILITY
```

See also

- “SYSCAPABILITY system view” on page 1349
- “SYSCAPABILITYNAME system view” on page 1349
SYSCATALOG consolidated view

Each row in the SYSCATALOG view describes a system table.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSCATALOG" ( creator, tname, dbspacename, tabletype, ncols, primary_key, "check", remarks )
    as select u.user_name, tab.table_name, dbs.dbspace_name,
        if tab.table_type_str = 'BASE' then 'TABLE' else tab.table_type_str endif,
        (select count() from SYS.ISYSTABCOL
         where ISYSTABCOL.table_id = tab.table_id),
        if ix.index_id is null then 'N' else 'Y' endif,
        null,
        rmk.remarks
    from SYS.SYSTAB as tab
    join SYS.ISYSDBSPACE as dbs on(tab.dbspace_id = dbs.dbspace_id)
    join SYS.ISYSUSER as u on u.user_id = tab.creator
    left outer join SYS.ISYSIDX as ix on(tab.table_id = ix.table_id and
        ix.index_id = 0)
    left outer join SYS.ISYSREMARK as rmk on(tab.object_id = rmk.object_id)
```

See also
- “SYSTAB system view” on page 1399
- “SYSTABLECOL system view” on page 1401
- “SYSDBSPACE system view” on page 1354
- “SYSUSER system view” on page 1412
- “SYSIDX system view” on page 1364
- “SYSREMARK system view” on page 1382

SYSCOLAUTH consolidated view

Each row in the SYSCOLAUTH view describes the set of privileges (UPDATE, SELECT, or REFERENCES) granted on a column. The SYSCOLAUTH view provides a user-friendly presentation of data in the “SYSCOLPERM system view” on page 1351.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSCOLAUTH" ( grantor, grantee, creator, tname, colname, privilege_type, is_grantable )
    as select u1.user_name, u2.user_name, u3.user_name, tab.table_name,
        col.column_name, cp.privilege_type, cp.is_grantable
    from SYS.ISYSCOLPERM as cp
    join SYS.ISYSUSER as u1 on u1.user_id = cp.grantor
    join SYS.ISYSUSER as u2 on u2.user_id = cp.grantee
    join SYS.ISYSUSER as u3 on u3.user_id = tab.creator
    join SYS.ISYSTAB as tab on tab.table_id = cp.table_id
    join SYS.ISYSTABCOL as col on col.table_id = cp.table_id
    and col.column_id = cp.column_id
```
SYSCOLSTATS consolidated view

The SYSCOLSTATS view contains the column statistics that are stored as histograms and used by the optimizer.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCOLSTATS" AS SELECT u.user_name, t.table_name, c.column_name, s.format_id, dateadd(mi, PROPERTY('TimeZoneAdjustment'), s.update_time) as update_time, s.density, s.max_steps, s.actual_steps, s.step_values, s.frequencies, TODATETIMEOFFSET(s.update_time, 0) as update_time_utc FROM SYS.ISYSCOLSTAT s JOIN SYS.ISYSTABCOL c on (s.table_id = c.table_id and s.column_id = c.column_id) JOIN SYS.ISYSTAB t on (t.table_id = c.table_id) JOIN SYS.ISYSUSER u on (u.user_id = t.creator)
```

See also
- “SYSCOLPERM system view” on page 1351
- “SYSTABCOL system view” on page 1401
- “SYSUSER system view” on page 1412
- “SYSTAB system view” on page 1399

SYSCOLUMNS consolidated view

Each row in the SYSCOLUMNS view describes one column of each table and view in the catalog.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSCOLUMNS"( creator,cname,tname,coltype,nulls,length, syslength,in_primary_key,colno,default_value, column_kind,remarks ) as select u.user_name,col.column_name,tab.table_name,dom.domain_name, col.nulls,col.width,col.scale,if ixcol.sequence is null then 'N' else 'Y' endif, col.column_id, col."default",col.column_type,rmk.remarks from SYS.SYSTABCOL as col left outer join SYS.ISYSIDXCOL as ixcol on(col.table_id = ixcol.table_id and col.column_id = ixcol.column_id and ixcol.index_id = 0) join SYS.ISYSTAB as tab on(tab.table_id = col.table_id) join SYS.ISYSDOMAIN as dom on(dom.domain_id = col.domain_id) join SYS.ISYSUSER as u on u.user_id = tab.creator left outer join SYS.ISYSREMARK as rmk on(col.object_id = rmk.object_id)
```

See also
- “SYSCOLSTAT system view” on page 1351
- “SYSTABCOL system view” on page 1401
- “SYSTAB system view” on page 1399
- “SYSUSER system view” on page 1412
SYSFOREIGNKEYS consolidated view

Each row in the SYSFOREIGNKEYS view describes one foreign key for each table in the catalog.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSFOREIGNKEYS" (foreign_creator,
   foreign_tname,
   primary_creator, primary_tname, role, columns)
AS SELECT fk_up.user_name, fk_tab.table_name, pk_up.user_name,
   pk_tab.table_name, ix.index_name,
   (SELECT list(string(fk_col.column_name, ' IS ', pk_col.column_name),'
    order by fkc.table_id, fkc.index_id, fkc."sequence")
   FROM SYS.ISYSIDXCOL AS fkc
   JOIN SYS.ISYSTABCOL AS fk_col ON (fkc.table_id = fk_col.table_id
   AND fkc.column_id = fk_col.column_id)
   JOIN SYS.ISYSTAB AS pk_col
   WHERE fkc.table_id = fk.foreign_table_id
   AND fkc.index_id = fk.foreign_index_id
   AND pk_col.table_id = fk.primary_table_id
   AND pk_col.column_id = fkc.primary_column_id)
FROM SYS.ISYSFKEY AS fk
JOIN SYS.ISYSTAB AS fk_tab ON fk_tab.table_id = fk.foreign_table_id
JOIN SYS.ISYSUSER AS fk_up ON fk_up.user_id = fk_tab.creator
JOIN SYS.ISYSTAB AS pk_tab ON pk_tab.table_id = fk.primary_table_id
JOIN SYS.ISYSUSER AS pk_up ON pk_up.user_id = pk_tab.creator
JOIN SYS.ISYSIDX AS ix ON ix.table_id = fk.foreign_table_id and
   ix.index_id = fk.foreign_index_id
```

See also

- “SYSTAB system view” on page 1399
- “SYSTABCOL system view” on page 1401
- “SYSIDX system view” on page 1364
- “SYSIDXCOL system view” on page 1366
- “SYSFKEY system view” on page 1361
- “SYSUSER system view” on page 1412
- “SYSDOMAIN system view” on page 1356
- “SYSREMARK system view” on page 1382
SYSINDEXES consolidated view

Each row in the SYSINDEXES view describes one index in the database. As an alternative to this view, you could also use the SYSIDX and SYSIDXCOL system views.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSINDEXES"( icreator,
iname,fname,creator,tname,indextype,
colnames,interval,level_num )
as select u.user_name,idx.index_name,dbs.dbspace_name,u.user_name,
tab.table_name,
case idx.index_category
when 1 then 'Primary Key'
when 2 then 'Foreign Key'
when 3 then
if idx."unique" = 4 then 'Non-unique'
else if idx."unique" = 2 then 'UNIQUE constraint'
else if idx."unique" = 5 then 'UNIQUE NULLS NOT DISTINCT'
else 'UNIQUE'
endif
endif
elseif
endif
when 4 then 'Text Index'
end,
(select list(string(c.column_name,
if ixc."order" = 'A' then ' ASC' else ' DESC' endif) order by
ixc.table_id asc,ixc.index_id asc,ixc.sequence asc)
from SYS.ISYSIDXCOL as ixc
join SYS.ISYSTABCOL as c on(
c.table_id = ixc.table_id
and c.column_id = ixc.column_id)
where ixc.index_id = idx.index_id
and ixc.table_id = idx.table_id),
0,0
from SYS.ISYSTAB as tab
join SYS.ISYSDBSPACE as dbs on(tab.dbspace_id = dbs.db_space_id)
join SYS.ISYSIDX as idx on(idx.table_id = tab.table_id)
join SYS.ISYSUSER as u on u.user_id = tab.creator
```

See also

- “SYSIDX system view” on page 1364
- “SYSTABCOL system view” on page 1401
- “SYSTAB system view” on page 1399
- “SYSDBSPACE system view” on page 1354
- “SYSIDXCOL system view” on page 1366
- “SYSUSER system view” on page 1412

SYSOPTIONS consolidated view

Each row in the SYSOPTIONS view describes one option created using the SET command. Each user can have their own setting for each option. In addition, settings for the PUBLIC user define the default settings to be used for users that do not have their own setting.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
```
ALTER VIEW "SYS"."SYSOPTIONS"( user_name,"option",setting )
as select u.user_name,opt."option",opt.setting
from SYS.ISYSOPTION as opt
  join SYS.ISYSUSER as u on opt.user_id = u.user_id

See also
- “SYSOPTION system view” on page 1375
- “SYSUSER system view” on page 1412

SYSPROCAUTH consolidated view
Each row in the SYSPROCAUTH view describes a set of privileges granted on a procedure. As an alternative, you can also use the SYSPROCPERM system view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

ALTER VIEW "SYS"."SYSPROCAUTH"( grantee, creator,procname )
as select u1.user_name,u2.user_name,p.proc_name
from SYS.ISYSPROCEDURE as p
  join SYS.ISYSPROCPERM as pp on(p.proc_id = pp.proc_id)
  join SYS.ISYSUSER as u1 on u1.user_id = pp.grantee
  join SYS.ISYSUSER as u2 on u2.user_id = p.creator

See also
- “SYSPROCEDURE system view” on page 1377
- “SYSPROCPERM system view” on page 1380
- “SYSUSER system view” on page 1412

SYSPROCPARMS consolidated view
Each row in the SYSPROCPARMS view describes a parameter to a procedure in the database.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

ALTER VIEW "SYS"."SYSPROCPARMS"( creator, procname,paramname,param_id,paramtype,parmmode,parmdomain,
  length,scale,"default",user_type )
as select up.user_name,p.proc_name,pp.parm_name,pp.parm_id,pp.parm_type,
  if pp.parm_mode_in = 'Y' and pp.parm_mode_out = 'N' then 'IN'
  else if pp.parm_mode_in = 'N' and pp.parm_mode_out = 'Y' then 'OUT'
  else 'INOUT'
  endif,dom.domain_name,pp.width,pp.scale,pp."default",ut.type_name
from SYS.SYSPROCPARM as pp
  join SYS.ISYSPROCEDURE as p on p.proc_id = pp.proc_id
  join SYS.ISYSUSER as up on up.user_id = p.creator
  join SYS.ISYSDOMAIN as dom on dom.domain_id = pp.domain_id
  left outer join SYS.ISYSUSERTYPE as ut on ut.type_id = pp.user_type
See also

- “SYSPROCPARM system view” on page 1378
- “SYSPROCEDURE system view” on page 1377
- “SYSUSER system view” on page 1412
- “SYSDOMAIN system view” on page 1356
- “SYSUSERTYPE system view” on page 1415

**SYSPROCS consolidated view**

The SYSPROCS view shows the procedure or function name, the name of its creator and any comments recorded for the procedure or function.

The tables and columns that make up this view are provided in the ALTER VIEW statement below.

```
ALTER VIEW "SYS"."SYSPROCS" ( creator, 
                     procname, remarks )
as select u.user_name, p.proc_name, r.remarks 
from SYS.ISYSPROCEDURE as p 
  join SYS.ISYSUSER as u on u.user_id = p.creator 
  left outer join SYS.ISYSREMARK as r on(p.object_id = r.object_id)
```

See also

- “SYSPROCEDURE system view” on page 1377
- “SYSUSER system view” on page 1412
- “SYSREMARK system view” on page 1382

**SYSPUBLICATIONS consolidated view**

Each row in the SYSPUBLICATIONS view describes a SQL Remote or MobiLink publication.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSPUBLICATIONS" 
as select u.user_name as creator, 
 p.publication_name, 
 r.remarks, 
 p.type, 
 case p.sync_type 
when 0 then 'logscan' 
when 1 then 'scripted upload' 
when 2 then 'download only' 
else 'invalid' 
end as sync_type 
from SYS.ISYSPUBLICATION as p 
  join SYS.ISYSUSER as u on u.user_id = p.creator 
  left outer join SYS.ISYSREMARK as r on(p.object_id = r.object_id)
```

See also

- “SYSPUBLICATION system view” on page 1381
- “SYSREMARK system view” on page 1382
SYSREMOOTEOPTION2 consolidated view

Joins together, and presents in a more readable format, the columns from SYSREMOOTEOPTION and SYSREMOOTEOPTIONTYPE system views.

Values in the setting column are hidden from users that do not have the SELECT ANY TABLE system privilege.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOOTEOPTION2"
    as select ISYSREMOOTEOPTION.option_id,
            ISYSREMOOTEOPTION.user_id,
            SYS.HIDE_FROM_NON_DBA(ISYSREMOOTEOPTION.setting) as setting
    from SYS.ISYSREMOOTEOPTION
```

See also

- “SYSREMOOTEOPTION system view” on page 1383

SYSREMOOTEOPTIONS consolidated view

Each row of the SYSREMOOTEOPTIONS view describes the values of a SQL Remote message link parameter.

Values in the setting column are hidden from users that do not have the SELECT ANY TABLE system privilege. The SYSREMOOTEOPTION2 view provides public access to the insensitive data.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOOTEOPTIONS"
    as select srt.type_name,
            sup.user_name,
            srot."option",
            SYS.HIDE_FROM_NON_DBA(sro.setting) as setting
    from SYS.ISYSREMOOTETYPE as srt
    ,SYS.ISYSREMOOTEOPTIONTYPE as srot
    ,SYS.ISYSREMOOTEOPTION as sro
    ,SYS.ISYSUSER as sup
    where srt.type_id = srot.type_id
    and srot.option_id = sro.option_id
    and sro.user_id = sup.user_id
```

See also

- “SYSREMOOTETYPE system view” on page 1383
- “SYSREMOOTEOPTIONTYPE system view” on page 1383
- “SYSREMOOTEOPTION system view” on page 1383
- “SYSUSER system view” on page 1412
SYSREMOTETYPES consolidated view

Each row of the SYSREMOTETYPES view describes one of the SQL Remote message types, including
the publisher address.

The tables and columns that make up this view are provided in the SQL statement below. To learn more
about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTETYPES"
    as select rt.type_id,rt.type_name,rt.publisher_address,rm.remarks
    from SYS.ISYSREMOTETYPE as rt
    left outer join SYS.ISYSREMARK as rm on(rt.object_id = rm.object_id)
```

See also
- “SYSREMOTETYPE system view” on page 1383
- “SYSREMARK system view” on page 1382

SYSREMOTEUSERS consolidated view

Each row of the SYSREMOTEUSERS view describes a user ID with the REMOTE system privilege (a
subscriber), together with the status of SQL Remote messages that were sent to and from that user.

The tables and columns that make up this view are provided in the SQL statement below. To learn more
about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSREMOTEUSERS" AS SELECT u.user_name, r.consolidate,
    t.type_name, r.address, r.frequency, r.send_time,
    (if r.frequency = 'A' then NULL
    else if r.frequency = 'P' then
        if r.time_sent IS NULL then CURRENT TIMESTAMP
        else (select min( minutes( dateadd(mi, PROPERTY('TimeZoneAdjustment'),
            a.time_sent),
            60*hour(a.send_time) + minute( seconds( a.send_time, 59 ) ) ) )
        FROM SYS.ISYSREMOTEUSER a WHERE a.frequency = 'P' AND a.send_time =
        r.send_time ) endif
    else if CURRENT DATE + r.send_time > coalesce( dateadd(mi,
        PROPERTY('TimeZoneAdjustment'), r.time_sent), CURRENT TIMESTAMP)
        then CURRENT DATE + r.send_time
    else CURRENT DATE + r.send_time + 1
        endif ) endif )
    as next_send, r.log_send ,
    dateadd(mi, PROPERTY('TimeZoneAdjustment'), r.time_sent)
    as time_sent ,
    r.log_received, r.confirm_received, r.receive_count, r.rereceive_count,
    TODATETIMEOFFSET( r.time_sent, 0 ) as time_sent_utc ,
    TODATETIMEOFFSET( r.time_received, 0 ) as time_received_utc
    FROM SYS.ISYSREMOTEUSER r JOIN SYS.ISYSUSER u ON ( u.user_id = r.user_id )
    JOIN SYS.ISYSREMOTETYPE t ON ( t.type_id = r.type_id )
```

See also
- “SYSREMOTEUSER system view” on page 1384
- “SYSUSER system view” on page 1412
- “SYSREMOTETYPE system view” on page 1383
SYSSUBSCRIPTIONS consolidated view

Each row describes a subscription from one user ID (which must have the REMOTE system privilege) to one publication.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSSUBSCRIPTIONS"
  as select p.publication_name,u.user_name,s.subscribe_by,s.created,
            s.started
  from SYS.ISYSSUBSCRIPTION as s
    join SYS.ISYSPUBLICATION as p on(p.publication_id = s.publication_id)
    join SYS.ISYSUSER as u on u.user_id = s.user_id
```

See also

- “SYSSUBSCRIPTION system view” on page 1395
- “SYSPUBLICATION system view” on page 1381
- “SYSUSER system view” on page 1412

SYSSYNC2 consolidated view

The SYSSYNC2 view provides public access to the data found in the SYSSYNC system view—information relating to MobiLink synchronization—without exposing potentially sensitive data.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

```
ALTER VIEW "SYS"."SYSSYNC2"
  as select ISYSSYNC.sync_id,
            ISYSSYNC.type,
            ISYSSYNC.publication_id,
            ISYSSYNC.progress,
            ISYSSYNC.site_name,
            SYS.HIDE_FROM_NON_DBA(ISYSSYNC."option") as "option",
            SYS.HIDE_FROM_NON_DBA(ISYSSYNC.server_connect) as server_connect,
            ISYSSYNC.server_conn_type,
            ISYSSYNC.last_download_time,
            ISYSSYNC.last_upload_time,
            ISYSSYNC.created,
            ISYSSYNC.log_sent,
            ISYSSYNC.generation_number,
            ISYSSYNC.extended_state,
            ISYSSYNC.script_version,
            ISYSSYNC.subscription_name
  from SYS.ISYSSYNC
```

See also

- “SYSSYNC system view” on page 1396
SYSSYNCPUBLICATIONDEFAULTS consolidated view

The SYSSYNCPUBLICATIONDEFAULTS view provides the default synchronization settings associated with publications involved in MobiLink synchronization.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

```sql
ALTER VIEW "SYS"."SYSSYNCPUBLICATIONDEFAULTS"
  as select s.sync_id,
          p.publication_name,
          SYS.HIDE_FROM_NON_DBA(s."option") as "option",
          SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
          s.server_conn_type
    from SYS.ISYSSYNC as s
       join SYS.ISYSPUBLICATION as p
         on (p.publication_id = s.publication_id)
    where s.site_name is null
```

See also
- “SYSSYNC system view” on page 1396
- “SYSPUBLICATION system view” on page 1381

SYSSYNCS consolidated view

The SYSSYNCS view contains information relating to MobiLink synchronization.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

```sql
ALTER VIEW "SYS"."SYSSYNCS"
  as select p.publication_name,s.progress,s.site_name,
          SYS.HIDE_FROM_NON_DBA(s."option") as "option",
          SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
          s.server_conn_type,s.last_download_time,
          s.last_upload_time,s.created,s.log_sent,s.generation_number,
          s.extended_state
    from SYS.ISYSSYNC as s
       left outer join SYS.ISYSPUBLICATION as p
         on (p.publication_id = s.publication_id)
```

See also
- “SYSSYNC system view” on page 1396
- “SYSPUBLICATION system view” on page 1381
SYSSYNCSCRIPTS consolidated view

Each row in the SYSSYNCSCRIPTS view identifies a stored procedure for MobiLink scripted upload. This view is almost identical to the SYSSYNCSCRIPT system view, except that the values are in human-readable format, as opposed to raw data.

```sql
ALTER VIEW "SYS"."SYSSYNCSCRIPTS"
 as select p.publication_name,
 t.table_name,
 case s.type
 when 0 then 'upload insert'
 when 1 then 'upload delete'
 when 2 then 'upload update'
 else 'unknown'
 end as type,
 c.proc_name
from SYS.ISYSSYNCSCRIPT as s
 join SYS.ISYSPUBLICATION as p on p.object_id = s.pub_object_id
 join SYS.ISYSTAB as t on t.object_id = s.table_object_id
 join SYS.ISYSPROCEDURE as c on c.object_id = s.proc_object_id
```

See also
- “SYSSYNCSCRIPT system view” on page 1398
- “SYSPUBLICATION system view” on page 1381
- “SYSTAB system view” on page 1399
- “SYSPROCEDURE system view” on page 1377
- “Scripted upload” [MobiLink - Client Administration]

SYSSYNCSUBSCRIPTIONS consolidated view

The SYSSYNCSUBSCRIPTIONS view contains the synchronization settings associated with MobiLink synchronization subscriptions.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

```sql
ALTER VIEW "SYS"."SYSSYNCSUBSCRIPTIONS"
 as select s.sync_id,
 p.publication_name,
 s.progress,
 s.site_name,
 SYS.HIDE_FROM_NON_DBA(s."option") as "option",
 SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
 s.server_conn_type,
 s.last_download_time,
 s.last_upload_time,
 s.created,
 s.log_sent,
 s.generation_number,
 s.extended_state
from SYS.ISYSSYNC as s join SYS.ISYSPUBLICATION as p on(p.publication_id = s.publication_id)
```

where s.publication_id is not null and
s.site_name is not null and exists
(select 1 from SYS.SYSSYNCUSERS as u
 where s.site_name = u.site_name)

See also

● “SYSSYNC system view” on page 1396
● “SYSPUBLICATION system view” on page 1381
● “SYSSYNCUSERS consolidated view” on page 1436

SYSSYNCUSERS consolidated view
A view of synchronization settings associated with MobiLink synchronization users.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

Values in the server_connect and option columns are hidden from users that do not have the SELECT ANY TABLE system privilege.

ALTER VIEW "SYS"."SYSSYNCUSERS"
as select ISYSSYNC.sync_id,
  ISYSSYNC.site_name,
  SYS.HIDE_FROM_NON_DBA(ISYSSYNC."option") as "option",
  SYS.HIDE_FROM_NON_DBA(ISYSSYNC.server_connect) as server_connect,
  ISYSSYNC.server_conn_type
from SYS.ISYSSYNC where
  ISYSSYNC.publication_id is null

See also

● “SYSSYNC system view” on page 1396

SYSTABAUTH consolidated view
The SYSTABAUTH view contains information from the SYSTABLEPERM system view, but in a more readable format.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

ALTER VIEW "SYS"."SYSTABAUTH"
  ( grantor,
    grantee,screator,stname,tcreator,ttname,
    selectauth,insertauth,deleteauth,
    updateauth,updatecols,alterauth,referenceauth,
    loadauth,truncateauth )
as select u1.user_name,u2.user_name,u3.user_name,tab1.table_name,
  u4.user_name,tab2.table_name,tp.selectauth,tp.insertauth,
  tp.deleteauth,tp.updateauth,tp.updatecols,tp.alterauth,
  tp.referenceauth,tp.loadauth,tp.truncateauth
from SYS.ISYSTABLEPERM as tp
  join SYS.ISYSUSER as u1 on u1.user_id = tp.grantor
  join SYS.ISYSUSER as u2 on u2.user_id = tp.grantee
  join SYS.ISYSTAB as tab1 on tab1.table_id = tp.stable_id
  join SYS.ISYSUSER as u3 on u3.user_id = tab1.creater
SYSTRIGGERS consolidated view

Each row in the SYSTRIGGERS view describes one trigger in the database. This view also contains triggers that are automatically created for foreign key definitions which have a referential triggered action (such as ON DELETE CASCADE).

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSTRIGGERS"( owner,
    trigname,tname,event,trigtime,trigdefn )
    as select u.user_name,trig.trigger_name,tab.table_name,
    if trig.event = 'I' then 'INSERT'
    else if trig.event = 'U' then 'UPDATE'
    else if trig.event = 'D' then 'DELETE'
    else if trig.event = 'A' then 'INSERT,DELETE'
    else if trig.event = 'B' then 'INSERT,UPDATE'
    else if trig.event = 'E' then 'DELETE,UPDATE'
    else 'INSERT,DELETE,UPDATE'
    endif
    endif
    endif
    endif
    endif,if trig.trigger_time = 'B' or trig.trigger_time = 'P' then 'BEFORE'
    else if trig.trigger_time = 'A' or trig.trigger_time = 'S' then 'AFTER'
    else if trig.trigger_time = 'R' then 'RESOLVE'
    else 'INSTEAD OF'
    endif
    endif
    endif
    endif
    endif,if trig.trigger_defn
from SYS.ISYSTRIGGER as trig
    join SYS.ISYSTAB as tab on(tab.table_id = trig.table_id)
    join SYS.ISYSUSER as u on u.user_id = tab.creator where
    trig.foreign_table_id is null
```

See also

- “SYSTRIGGER system view” on page 1409
- “SYSTAB system view” on page 1399
- “SYSUSER system view” on page 1412
SYSUSEROPTIONS consolidated view

The SYSUSEROPTIONS view contains the option settings that are in effect for each user. If a user has no setting for an option, this view displays the public setting for the option.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSUSEROPTIONS"( user_name, 
    "option",setting )
    as select u.user_name,
            o."option",
            isnull((select s.setting 
                from SYS.ISYSOPTION as s 
                where s.user_id = u.user_id 
                and s."option" = o."option"), 
            o.setting)
    from SYS.SYSOPTIONS as o,SYS.ISYSUSER as u 
    where o.user_name = 'PUBLIC'
```

See also

- “SYSOPTIONS consolidated view” on page 1428
- “SYSUSER system view” on page 1412

SYSVIEWS consolidated view

Each row of the SYSVIEWS view describes one view, including its view definition.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSVIEWS"( vcreator, 
    viewname,viewtext )
    as select u.user_name,t.table_name,v.view_def
    from SYS.ISYSTAB as t 
        join SYS.ISYSVIEW as v on(t.object_id = v.view_object_id) 
        join SYS.ISYSUSER as u on(u.user_id = t.creator)
```

See also

- “SYSTAB system view” on page 1399
- “SYSVIEW system view” on page 1416
- “SYSUSER system view” on page 1412

Compatibility views

Compatibility views are views that are provided for compatibility with versions of SQL Anywhere 10 and earlier. Where possible you should use system and consolidated views instead, as support may diminish for some compatibility views in future releases.
SYSCOLLATION compatibility view (deprecated)

The SYSCOLLATION compatibility view contains the collation sequence information for the database. It is obtainable via built-in functions and is not kept in the catalog. Following is definition for this view:

```sql
ALTER VIEW "SYS"."SYSCOLLATION"
as select 1 as collation_id,
   DB_PROPERTY('Collation') as collation_label,
   DB_EXTENDED_PROPERTY('Collation','Description') as collation_name,
   cast(DB_EXTENDED_PROPERTY('Collation','LegacyData') as binary(1280)) as collation_order
```

See also
- “List of database properties” [SQL Anywhere Server - Database Administration]
- “DB_PROPERTY function [System]” on page 218
- “DB_EXTENDED_PROPERTY function [System]” on page 212

SYSCOLLATIONMAPPINGS compatibility view (deprecated)

The SYSCOLLATIONMAPPINGS compatibility view contains only one row with the database collation mapping. It is obtainable via built-in functions and is not kept in the catalog. Following is definition for this view:

```sql
ALTER VIEW "SYS"."SYSCOLLATIONMAPPINGS"
as select DB_PROPERTY('Collation') as collation_label,
   DB_EXTENDED_PROPERTY('Collation','Description') as collation_name,
   DB_PROPERTY('Charset') as cs_label,
   DB_EXTENDED_PROPERTY('Collation','ASESensitiveSortOrder') as so_case_label,
   DB_EXTENDED_PROPERTY('Collation','ASEInsensitiveSortOrder') as so_caseless_label,
   DB_EXTENDED_PROPERTY('Charset','java') as jdk_label
```

See also
- “List of database properties” [SQL Anywhere Server - Database Administration]
- “DB_PROPERTY function [System]” on page 218
- “DB_EXTENDED_PROPERTY function [System]” on page 212

SYSCOLUMN compatibility view (deprecated)

The SYSCOLUMN view is provided for compatibility with older versions of the software that offered a SYSCOLUMN system table. However, the previous SYSCOLUMN table has been replaced by the ISYSTABCOL system table, and its corresponding SYSTABCOL system view, which you should use instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSCOLUMN"
as select b.table_id,
```
b.column_id,
if c.sequence is null then 'N' else 'Y' endif as pkey,
b.domain_id,
b.nulls,
b.width,
b.scale,
b.object_id,
b.max_identity,
b.column_name,
r.remarks,
b."default",
b.user_type,
b.column_type
from SYS.SYSTABCOL as b
left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
left outer join SYS.ISYSIDXCOL as c on(b.table_id = c.table_id and
b.column_id = c.column_id and c.index_id = 0)

See also

- “SYSTABCOL system view” on page 1401
- “SYSREMARK system view” on page 1382
- “SYSIDXCOL system view” on page 1366

SYSFILE compatibility view (deprecated)

Each row in the SYSFILE system view describes a dbspace for a database. Every database consists of one or more dbspaces; each dbspace corresponds to an operating system file.

dbspaces are automatically created for the main database file, temporary file, transaction log file, and transaction log mirror file. Information about the transaction log, and transaction log mirror dbspaces does not appear in the SYSFILE system view.

ALTER VIEW "SYS"."SYSFILE"
as select b.dbfile_id as file_id,
if b.dbspace_id = 0 and b.dbfile_id = 0 then
db_property('File')
else
  if b.dbspace_id = 15 and b.dbfile_id = 15 then
db_property('TempFileName')
  else
    b.file_name
  endif
endif as file_name,
a.dbspace_name,
a.store_type,
b.lob_map,
b.dbspace_id
from SYS.ISYSDBSPACE as a
join SYS.ISYSDBFILE as b on(a.dbspace_id = b.dbspace_id)

See also

- “Predefined dbspaces” [SQL Anywhere Server - Database Administration]
### SYSFKCOL compatibility view (deprecated)

Each row of SYSFKCOL describes the association between a foreign column in the foreign table of a relationship and the primary column in the primary table. This view is deprecated; use the SYSIDX and SYSIDXCOL system views instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSFKCOL"
 as select a.table_id as foreign_table_id,
         a.index_id as foreign_key_id,
         a.column_id as foreign_column_id,
         a.primary_column_id
from SYS.ISYSIDXCOL as a
 ,SYS.ISYSIDX as b
where a.table_id = b.table_id
and a.index_id = b.index_id
and b.index_category = 2
```

See also
- “SYSIDX system view” on page 1364
- “SYSIDXCOL system view” on page 1366

### SYSFOREIGNKEY compatibility view (deprecated)

The SYSFOREIGNKEY view is provided for compatibility with older versions of the software that offered a SYSFOREIGNKEY system table. However, the previous SYSFOREIGNKEY system table has been replaced by the ISYSFKKEY system table, and its corresponding SYSFKKEY system view, which you should use instead.

A foreign key is a relationship between two tables—the foreign table and the primary table. Every foreign key is defined by one row in SYSFOREIGNKEY and one or more rows in SYSFKCOL. SYSFOREIGNKEY contains general information about the foreign key while SYSFKCOL identifies the columns in the foreign key and associates each column in the foreign key with a column in the primary key of the primary table.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSFOREIGNKEY"
 as select b.foreign_table_id,
         b.foreign_index_id as foreign_key_id,
         a.object_id,
         b.primary_table_id,
         p.root,
         b.check_on_commit,
         b.nulls,
         a.index_name as role,
         r.remarks,
         b.primary_index_id,
         a.not_enforced as fk_not_enforced,
         10 as hash_limit
from SYS.ISYSIDX as a left outer join SYS.ISYSPHYSIDX as p
on(a.table_id
SYSGROUP compatibility view

There is one row in the SYSGROUP system view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>UNSIGNED INT</td>
<td>The user number of the group.</td>
</tr>
<tr>
<td>group_member</td>
<td>UNSIGNED INT</td>
<td>The user number of a member.</td>
</tr>
</tbody>
</table>

Constraints on underlying system table

- PRIMARY KEY (group_id, group_member)
- FOREIGN KEY group_id (group_id) REFERENCES SYS.ISYSUSER (user_id)
- FOREIGN KEY group_member (group_member) REFERENCES SYS.ISYSUSER (user_id)

SYSGROUPS compatibility view

There is one row in the SYSGROUPS view for each member of each group. This view describes the many-to-many relationship between groups and members. A group may have many members, and a user may be a member of many groups.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSGROUPS"( group_name,
member_name )
as select g.user_name,u.user_name
from SYS.ISYSROLEGRANT,SYS.ISYSUSER as g,SYS.ISYSUSER as u
where ISYSROLEGRANT.role_id = g.user_id and ISYSROLEGRANT.grantee =
u.user_id and( u.user_name in( 'SYS_SPATIAL_ADMIN_ROLE' ) or u.user_id <= 2147483648) and( g.user_type = (0x02|0x04|0x08) or g.user_name in( 'SYS','PUBLIC','dbo','diagnostics','rs_systabgroup','SA_DEBUG','SYS_SPATIAL_ADMIN_ROLE' ) )
```


See also

- “SYSUSER system view” on page 1412
- “SYSGROUP compatibility view” on page 1442

SYSINDEX compatibility view (deprecated)

The SYSINDEX view is provided for compatibility with older versions of the software that offered a SYSINDEX system table. However, the SYSINDEX system table has been replaced by the ISYSIDX system table, and its corresponding SYSIDX system view, which you should use instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSINDEX"
as select b.table_id,
b.index_id,
b.object_id,
p.root,
b.dbspace_id,
case b."unique"
when 1 then 'Y'
when 2 then 'U'
when 3 then 'M'
when 4 then 'N'
when 5 then 'Y'
else 'I'
end as "unique",
t.creator,
b.index_name,
r.remarks,
10 as hash_limit,
b.dbspace_id as file_id
from(SYS.ISYSIDX as b left outer join SYS.ISYSPHYSIDX as p on(b.table_id = p.table_id and b.phys_index_id = p.phys_index_id))
left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
,SYS.ISYSTAB as t
where t.table_id = b.table_id
and b.index_category = 3
```

See also

- “SYSIDX system view” on page 1364
- “SYSPHYSIDX system view” on page 1376
- “SYSTABLE compatibility view (deprecated)” on page 1444
- “SYSREMARK system view” on page 1382

SYSINFO compatibility view (deprecated)

The SYSINFO view indicates the database characteristics, as defined when the database was created. It always contains only one row. This view is obtainable via built-in functions and is not kept in the catalog. Following is the definition for the SYSINFO view:

```
ALTER VIEW "SYS"."SYSINFO"( page_size,
encryption,
```
blank_padding,
case_sensitivity,
default_collation,
database_version )
as select db_property('PageSize'),
    if db_property('Encryption') <> 'None' then 'Y' else 'N' endif,
    if db_property('BlankPadding') = 'On' then 'Y' else 'N' endif,
    if db_property('CaseSensitive') = 'On' then 'Y' else 'N' endif,
    db_property('Collation'),
    NULL

See also
● “List of database properties” [SQL Anywhere Server - Database Administration]
● “DB_PROPERTY function [System]” on page 218
● “DB_EXTENDED_PROPERTY function [System]” on page 212

SYSIXCOL compatibility view (deprecated)
The SYSIXCOL view is provided for compatibility with older versions of the software that offered a
SYSIXCOL system table. However, the SYSIXCOL system table has been replaced by the
ISYSIDXCOL system table, and its corresponding SYSIDXCOL system view. You should switch to
using the SYSIDXCOL system view.

Each row of the SYSIXCOL describes a column in an index. The tables and columns that make up this
view are provided in the SQL statement below. To learn more about a particular table or column, use the
links provided beneath the view definition.

```
ALTER VIEW "SYS"."SYSIXCOL"
as select a.table_id,
a.index_id,
a.sequence,
a.column_id,
a."order"
from SYS.ISYSIDXCOL as a
, SYS.ISYSIDX as b
where a.table_id = b.table_id
and a.index_id = b.index_id
and b.index_category = 3
```

See also
● “SYSIDX system view” on page 1364
● “SYSIDXCOL system view” on page 1366

SYSTABLE compatibility view (deprecated)
The SYSTABLE view is provided for compatibility with older versions of the software that offered a
SYSTABLE system table. However, the SYSTABLE system table has been replaced by the ISYSTAB
system table, and its corresponding SYSTAB system view, which you should use instead.

Each row of SYSTABLE view describes one table in the database.
The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSTABLE"
    as select b.table_id,
            b.file_id,
            b.count,
          0 as first_page,
            b.commit_action as last_page,
        COALESCE(ph.root,0) as primary_root,
            b.creator,
          0 as first_ext_page,
          0 as last_ext_page,
            b.table_page_count,
            b.ext_page_count,
            b.object_id,
            b.table_name,
            b.table_type_str as table_type,
            v.view_def,
            r.remarks,
            b.replicate,
            p.existing_obj,
            p.remote_location,
            'T' as remote_objtype,
            p.srvid,
   case b.server_type
     when 1 then 'SA'
     when 2 then 'IQ'
     when 3 then 'OMNI'
     else 'INVALID'
   end as server_type,
      10 as primary_hash_limit,
      0 as page_map_start,
      s.source,
            b."encrypted"

from SYS.SYSTAB as b
    left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
    left outer join SYS.ISYSSOURCE as s on(b.object_id = s.object_id)
    left outer join SYS.ISYSVIEW as v on(b.object_id = v.view_object_id)
    left outer join SYS.ISYSPROXYTAB as p on(b.object_id = p.table_object_id)
    left outer join(SYS.ISYSISDX as i left outer join SYS.ISYSPHYSIDX as ph
    on(i.table_id = ph.table_id
      and i.phys_index_id = ph.phys_index_id)) on(b.table_id = i.table_id
    and i.index_category = 1
      and i.index_id = 0)
```

See also

- “SYSTAB system view” on page 1399
- “SYSREMARK system view” on page 1382
- “SYSSOURCE system view” on page 1391
- “SYSVIEW system view” on page 1416
- “SYSPROXYTAB system view” on page 1380
- “SYSIDX system view” on page 1364
- “SYSPHYSIDX system view” on page 1376
SYSUSERAUTH compatibility view (deprecated)

Each row of the SYSUSERAUTH view describes a user, without exposing their user_id. Instead, each user is identified by their user name. Because this view displays passwords, you must have the SELECT ANY TABLE system privilege to view its data.

The SYSUSERAUTH view is provided for compatibility with older versions of the software. Use the SYSROLEGRANTS consolidated view instead.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

**Note**
Although the title of this view contains the word auth (for authorities), the security model is based on roles and privileges. The data in the view is therefore compiled using role information from the tables and views mentioned in the view definition.

```
ALTER VIEW "SYS"."SYSUSERAUTH"( name, password, resourceauth, dbaauth, scheduleauth, user_group )
    as select
    SYSUSERPERM.user_name, SYSUSERPERM.password, SYSUSERPERM.resourceauth, SYSUSERPERM.dbaauth, SYSUSERPERM.scheduleauth, SYSUSERPERM.user_group
    from SYS.SYSUSERPERM
```

See also
- “SYSROLEGRANTS consolidated view” on page 1387
- “SYSROLEGRANT system view” on page 1386
- “SYSROLEGRANTEXT system view” on page 1387

SYSUSERAUTHORITY compatibility view (deprecated)

The SYSUSERAUTHORITY view is provided for compatibility with older versions of the software. Use the SYSROLEGRANTS consolidated view instead.

Each row of SYSUSERAUTHORITY system view describes an authority granted to one user ID.

**Note**
Although the title of this view contains the word authority, the security model is based on roles and privileges. The data in the view is therefore compiled using role information from the tables and views mentioned in the view definition.

```
ALTER VIEW "SYS"."SYSUSERAUTHORITY" as
    select ISYSROLEGRANT.grantee as user_id,
        sp_auth_sys_role_info.auth
    from SYS.ISYSROLEGRANT
        natural join dbo.sp_auth_sys_role_info()
    where ISYSROLEGRANT.grant_type <> (0x02 | 0x04) and
        not ISYSROLEGRANT.grantee = any(select sp_auth_sys_role_info.role_id
            from dbo.sp_auth_sys_role_info())
    union
    select ISYSUSER.user_id,
        cast('GROUP' as varchar(20)) as auth
```

1446Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
from SYS.ISYSUSER
where ISYSUSER.user_name
in( 'SYS','PUBLIC','diagnostics','SYS_SPATIAL_ADMIN_ROLE','rs_systabgroup','S
A_DEBUG','dbo') union
select ISYSUSER.user_id,
cast('GROUP' as varchar(20)) as auth
from SYS.ISYSUSER
where ISYSUSER.user_type = (0x02|0x04|0x08) union
select cast(opt.setting as unsigned integer) as user_id,
cast('PUBLISH' as varchar(20)) as auth
from SYS.ISYSOPTION as opt
where opt."option" like '%db_publisher%' and opt.setting not like '%-%-1%'

See also
- “SYSROLEGRANTS consolidated view” on page 1387
- “SYSROLEGRANT system view” on page 1386
- “SYSROLEGRANTTEXT system view” on page 1387

SYSUSERLIST compatibility view (deprecated)
The SYSUSERAUTH view is provided for compatibility with older versions of the software.

Each row of the SYSUSERLIST view describes a user, without exposing their user_id and password. Each user is identified by their user name.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSUSERLIST"( name,
   resourceauth,dbaauth,scheduleauth,user_group )
as select
SYSUSERPERM.user_name, SYSUSERPERM.resourceauth, SYSUSERPERM.dbaauth, SYSUSERPERM.scheduleauth, SYSUSERPERM.user_group
from SYS.SYSUSERPERM
```

See also
- “SYSUSERPERM compatibility view (deprecated)” on page 1447

SYSUSERPERM compatibility view (deprecated)
Each row of the SYSUSERPERM view describes one user ID. You must have the SELECT ANY TABLE system privilege to view data in this view.

This view is deprecated because it only shows the authorities and permissions available in previous versions. You should change your application to use the SYSROLEGRANTS consolidated view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSUSERPERM"
   as select b.user_id,
b.object_id,
```
b.user_name,
b.password,
if AA.resourceauth is not null and AA.resourceauth > 0 then
  'Y' else 'N' endif as resourceauth,
if AA.dbaauth is not null and AA.dbaauth > 0 then
  'Y' else 'N' endif as dbaauth,
'N' as scheduleauth,
if exists(select * from SYS.ISYSOPTION as opt
  where opt."option" like '%db_publisher%' and opt.setting not like '%-1'
  and b.user_id = cast(opt.setting as integer)) then
  'Y' else 'N' endif as publishauth,
if AA.remotedbaauth is not null and AA.remotedbaauth > 0 then
  'Y' else 'N' endif as remotedbaauth,
if b.user_type = (0x02|0x04|0x08) or b.user_name
  in( 'SYS','PUBLIC','diagnostics','SYS_SPATIAL_ADMIN_ROLE','rs_systabgroup','SA_DEBUG','dbo' ) then
  'Y' else 'N' endif as user_group,
r.remarks
from SYS.ISYSUSER as b
left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
left outer join(select sum(if sp_auth_sys_role_info.auth = 'RESOURCE'
  then 1 else 0 endif) as resourceauth,
  sum(if sp_auth_sys_role_info.auth = 'DBA' then 1 else 0 endif) as dbaauth,
  sum(if sp_auth_sys_role_info.auth = 'REMOTE DBA' then 1 else 0 endif) as remotedbaauth,
  ISYSROLEGRANT.grantee
  from SYS.ISYSROLEGRANT natural join dbo.sp_auth_sys_role_info()
  where ISYSROLEGRANT.grant_type <> (0x02|0x04)
  and sp_auth_sys_role_info.auth in( 'DBA','RESOURCE','REMOTE DBA' )
  group by ISYSROLEGRANT.grantee) as AA
on(AA.grantee = b.user_id)

See also
- “SYSROLEGRANTS consolidated view” on page 1387
- “SYSROLEGRANT system view” on page 1386
- “SYSROLEGRANTEXT system view” on page 1387
- “SYSUSER system view” on page 1412

SYSUSERPERMS compatibility view (deprecated)

This view is deprecated because it only shows the authorities and permissions available in previous versions. You should change your application to use the SYSROLEGRANTS consolidated view.

Each row of the SYSUSERPERMS view describes one user ID. However, password information is not included. All users are allowed to read from this view.

The tables and columns that make up this view are provided in the SQL statement below. To learn more about a particular table or column, use the links provided beneath the view definition.

```sql
ALTER VIEW "SYS"."SYSUSERPERMS"
  as select
  SYSUSERPERM.user_id, SYSUSERPERM.user_name, SYSUSERPERM.resourceauth, SYSUSERPERM.dbaauth,
  SYSUSERPERM.scheduleauth, SYSUSERPERM.user_group, SYSUSERPERM.publishauth, SYSUS...```
Views for Transact-SQL compatibility

The Adaptive Server Enterprise and SQL Anywhere system catalogs are different. The Adaptive Server Enterprise system tables and views are owned by the user dbo, and exist partly in the master database, partly in the sybsecurity database, and partly in each individual database. The SQL Anywhere system tables and views are owned by the special user SYS and exist separately in each database.

To assist in preparing compatible applications, SQL Anywhere provides the following set of views owned by the special user dbo, which correspond to their Adaptive Server Enterprise counterparts. Where architectural differences make the contents of a particular Adaptive Server Enterprise table or view meaningless in a SQL Anywhere context, the view is empty, containing just the column names and data types.

<table>
<thead>
<tr>
<th>View name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>syscolumns</td>
<td>One row for each column in a table or view, and for each parameter in a procedure.</td>
</tr>
<tr>
<td>syscomments</td>
<td>One or more rows for each view, rule, default, trigger, and procedure, giving the SQL definition statement.</td>
</tr>
<tr>
<td>sysindexes</td>
<td>One row for each clustered or nonclustered index, one row for each table with no indexes, and an additional row for each table containing text or image data.</td>
</tr>
<tr>
<td>sysobjects</td>
<td>One row for each table, view, procedure, rule, trigger default, log, or (in tempdb only) temporary object.</td>
</tr>
<tr>
<td>systypes</td>
<td>One row for each system-supplied or user-defined data type.</td>
</tr>
<tr>
<td>sysusers</td>
<td>One row for each user allowed in the database.</td>
</tr>
<tr>
<td>syslogins</td>
<td>One row for each valid user account.</td>
</tr>
</tbody>
</table>
Index

Symbols

!<
  comparison operator, 9
!=
  comparison operator, 9
!>
  comparison operator, 9
% comment indicator
  about, 85
% operator
  modulo function, 299
&
  bitwise operator, 19
- comment indicator
  about, 85
/* comment indicator
  about, 85
// comment indicator
  about, 85
0x
  binary literals, 6
<
  comparison operator, 9
<=
  comparison operator, 9
<>
  comparison operator, 9
=
  comparison operator, 9
>
  comparison operator, 9
>=
  comparison operator, 9
@@char_convert global variable
  about, 82
@@client_csid global variable
  about, 82
@@client_csname global variable
  about, 82
@@connections global variable
  about, 82
@@cpu_busy global variable
  about, 82
@@dbts global variable
  about, 82
@@error global variable
  about, 82
@@fetch_status global variable
  about, 82
@@identity global variable
  about, 82
description, 85
triggers, 85
@@idle global variable
  about, 82
@@io_busy global variable
  about, 82
@@isolation global variable
  about, 82
@@langid global variable
  about, 82
@@language global variable
  about, 82
@@max_connections global variable
  about, 82
@@maxcharlen global variable
  about, 82
@@ncharsize global variable
  about, 82
@@nestlevel global variable
  about, 82
@@pack_received global variable
  about, 82
@@pack_sent global variable
  about, 82
@@packet_errors global variable
  about, 82
@@procid global variable
  about, 82
@@rowcount global variable
  about, 82
@@servername global variable
  about, 82
@@spid global variable
  about, 82
@@sqlstatus global variable
  about, 82
@@textsize global variable
  about, 82
@@thresh_hysteresis global variable
  about, 82
@@timeticks global variable
Index

about, 82
@@total_errors global variable
about, 82
@@total_read global variable
about, 82
@@total_write global variable
about, 82
@@tranchained global variable
about, 82
@@trancount global variable
about, 82
@@transtate global variable
about, 82
@@version global variable
about, 82
@HttpMethod
HTTP_HEADER function, 267
@HttpGetQuery
HTTP_HEADER function, 267
@HttpGetStatus header
HTTP_RESPONSE_HEADER function, 269
sa_set_http_header_info system procedure, 1239
@HttpGetURI
HTTP_HEADER function, 267
@HttpGetVersion
HTTP_HEADER function, 267
@mp:id metaproperty
OPENXML operator, 13
@mp:localname metaproperty
OPENXML operator, 13
@mp:namespaceuri metaproperty
OPENXML operator, 13
@mp:prefix metaproperty
OPENXML operator, 13
@mp:xmltext metaproperty
OPENXML operator, 13
[ESQL]
statement indicators, 424
[Interactive SQL]
statement indicators, 424
[SP]
statement indicators, 424
[T-SQL]
statement indicators, 424
~
bitwise operator, 19
substituting for near expression, 57

A
ABS function
syntax, 155
ABSOLUTE clause
FETCH statement, 801
ACOS function
syntax, 156
Adaptive Server Enterprise
catalog procedures, 1091
converting stored procedures to Watcom SQL
syntax, 406
CREATE DATABASE statement, 550
migrating to SQL Anywhere using sa_migrate
system procedure, 1184
system procedures, 1090
ADD clause
ALTER DBSPACE statement, 433
ALTER PUBLICATION statement, 457
ALTER TABLE statement, 488
ADD OPTION clause
ALTER SYNCHRONIZATION SUBSCRIPTION
statement [MobiLink], 483
ALTER SYNCHRONIZATION USER statement
[MobiLink], 485
ADD PCTFREE clause
ALTER MATERIALIZED VIEW statement, 450
ADD table-constraint clause
ALTER TABLE statement, 493
ADD | ALTER | DELETE SCHEDULE clause
ALTER EVENT statement, 437
adding
certificates, 547
columns using the ALTER TABLE statement, 486
indexes using the CREATE INDEX statement, 599
Java classes, 868
LDAP server configuration objects, 603
messages, 614
servers, 657
ADDRESS clause
ALTER REMOTE MESSAGE TYPE statement, 459
ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 483
ALTER SYNCHRONIZATION USER statement [MobiLink], 485
CREATE PUBLICATION statement [MobiLink] [SQL Remote], 651
CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 687
CREATE SYNCHRONIZATION USER, 689
GRANT CONSOLIDATE statement [SQL Remote], 834
GRANT REMOTE statement [SQL Remote], 844
addresses
SQL Remote publishers, 459
AES encryption algorithm
CREATE DATABASE statement, 549
CREATE ENCRYPTED FILE statement, 567
DECRYPT function, 220
ENCRYPT function, 226
AES256 encryption algorithm
CREATE DATABASE statement, 549
CREATE ENCRYPTED FILE statement, 567
DECRYPT function, 220
ENCRYPT function, 226
AES256_FIPS encryption algorithm
CREATE DATABASE statement, 549
CREATE ENCRYPTED FILE statement, 567
DECRYPT function, 220
ENCRYPT function, 226
AES_FIPS encryption algorithm
CREATE DATABASE statement, 549
CREATE ENCRYPTED FILE statement, 567
DECRYPT function, 220
ENCRYPT function, 226
ALLOW clause
LOAD TABLE statement, 877
alphabetic characters
defined, 4
alphabetical list of binary data types
about, 124
alphabetical list of bit array data types
about, 107
alphabetical list of composite data types
about, 128
alphabetical list of date and time data types
about, 108
alphabetical list of money data types
about, 106
alphabetical list of numeric data types
about, 97
ALTER clause
ALTER TABLE statement, 494
ALTER DATABASE statement
FORCE START clause, 430
SET PARTNER FAILOVER clause, 428
syntax, 426
ALTER DATABASE UPGRADE statement
syntax, 426
ALTER DATATYPE statement
syntax, 435
ALTER DBSPACE statement
syntax, 432
ALTER DOMAIN statement
syntax, 435
ALTER EVENT statement
syntax, 436
ALTER EXTERNAL ENVIRONMENT statement
syntax, 438
ALTER FUNCTION statement
syntax, 439
ALTER INDEX statement
syntax, 441
ALTER LDAP SERVER statement
syntax, 442
ALTER LOGIN POLICY statement
syntax, 445
ALTER MATERIALIZED VIEW statement
syntax, 450
ALTER MIRROR SERVER statement
syntax, 452
ALTER OPTION clause
ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 483
ALTER SYNCHRONIZATION USER statement [MobiLink], 485
ALTER PARENT FROM clause
ALTER MIRROR SERVER statement, 454
ALTER privilege
GRANT statement, 828
REVOKE statement, 946
ALTER privilege clause
GRANT statement, 830
ALTER PROCEDURE statement
syntax, 456
ALTER PUBLICATION statement
MobiLink syntax, 457
SQL Remote syntax, 457
ALTER REMOTE MESSAGE TYPE statement
ALTER PUBLICATION statement, 457
ALTER role
ALTER ROLE statement
syntax, 460
ALTER SEQUENCE statement
syntax, 462
ALTER SERVER statement
syntax, 463
ALTER SERVICE statement [HTTP web service]
syntax, 466
ALTER SERVICE statement [SOAP web service]
syntax, 471
ALTER SPATIAL REFERENCE SYSTEM statement
syntax, 477
ALTER STATISTICS statement
syntax, 480
ALTER SYNCHRONIZATION PROFILE statement
MobiLink syntax, 481
ALTER SYNCHRONIZATION SUBSCRIPTION statement
MobiLink syntax, 482
ALTER SYNCHRONIZATION USER statement
MobiLink syntax, 485
ALTER TABLE OWNER clause
ALTER TABLE statement, 497
ALTER TABLE statement
syntax, 486
ALTER TEXT CONFIGURATION statement
syntax, 501
ALTER TEXT INDEX statement
syntax, 504
ALTER USER statement
character set conversion on passwords, 510
password maximum length, 510
passwords, character set conversion on, 510
syntax, 508
ALTER TRIGGER statement
syntax, 507
ALTER VIEW statement
DISABLE clause, 512
ENABLE clause, 512
RECOMPILE clause, 512
syntax, 511
altering
ALTER PUBLICATION statement, 457
ALTER TABLE statement, 486
columns using the ALTER TABLE statement, 486
data types using the ALTER DOMAIN statement, 435
databases using the ALTER DATABASE statement, 426
dbspaces using the ALTER DBSPACE statement, 432
domains using the ALTER DOMAIN statement, 435
events using the ALTER EVENT statement, 436
indexes using the ALTER INDEX statement, 441
LDAP server configuration objects, 442
login policy options using ALTER LOGIN POLICY statement, 445
login policy options using ALTER USER statement, 508
materialized views using the ALTER MATERIALIZED VIEW statement, 450
procedures using the ALTER PROCEDURE statement, 456
remote server attributes using the ALTER SERVER statement, 463
SQL Remote ALTER PUBLICATION statement, 457
SQL Remote remote message types, 459
text configuration objects, 501
text indexes using ALTER TEXT INDEX statement, 504
triggers using the ALTER TRIGGER statement, 507
views using the ALTER VIEW statement, 511
altering synchronization profiles
ALTER SYNCHRONIZATION PROFILE statement [MobiLink], 481
alternate server names
defining with CREATE MIRROR SERVER statement, 616
AND
bitwise operators, 19
logical operators description, 10
three-valued logic, 63
ANGULAR UNIT OF MEASURE clause
ALTER SPATIAL REFERENCE SYSTEM statement, 478
CREATE SPATIAL REFERENCE SYSTEM statement, 676
ANSI
equivalency using the REWRITE function, 348
ansi_nulls option
Microsoft SQL Server compatibility, 988
ansi_permissions option
setting with Transact-SQL SET statement, 987
ansinull option
setting with Transact-SQL SET statement, 987
ANY search condition
syntax, 43
apostrophes
in SQL strings, 8
APPEND clause
OUTPUT statement, 908
UNLOAD statement, 1028
application profiling
setting the tracing level, 1245
approximate data types
about, 97
arbiter servers
defining with CREATE MIRROR SERVER statement, 616
arc-cosine function
ACOS function, 156
arc-sine function
ASIN function, 161
arc-tangent function
ATAN function, 162
ATAN2 function, 163
archive backups
supported operating systems using the BACKUP statement, 516
archives
creating database backups using the BACKUP statement, 516
restoring databases from, 937
ARGN function
syntax, 157
arithmetic operators
Modulo, 11
SQL syntax, 11
ARRAY clause
EXECUTE statement, 794
FETCH statement, 802
PUT statement [ESQL], 919
ARRAY constructor
syntax, 157
ARRAY data type
declaring, 128
array operators
syntax, 18
UNNEST operator, 18
ARRAY_AGG function
syntax, 159
ARRAY_MAX_CARDINALITY function
syntax, 160
arrays (see composite data types)
articles
SYSARTICLE system view, 1348
SYSARTICLECOL system view, 1348
AS clause
ALTER MIRROR SERVER statement, 453
ALTER SERVICE statement [HTTP web service], 466
ALTER SERVICE statement [SOAP web service], 471
ALTER VIEW statement, 511
CONNECT statement [ESQL] [Interactive SQL], 544
CREATE DBSPACE statement, 557
CREATE DOMAIN statement, 562
CREATE FUNCTION statement, 594
CREATE MATERIALIZED VIEW statement, 612
CREATE MESSAGE statement, 614
CREATE MIRROR SERVER statement, 616
CREATE PROCEDURE statement [T-SQL], 637
CREATE TRIGGER [Transact-SQL] statement, 720
CREATE VIEW statement, 725
DELETE statement, 741
START DATABASE statement, 993
UPDATE statement, 1037
ASC | DESC clause
CREATE INDEX statement, 600
ASCII
function and syntax, 161
ASE COMPATIBLE clause
CREATE DATABASE statement, 550
ASIN function
syntax, 161
assertions
regular expression examples, 34
regular expressions, 34
assigning
logins for remote servers, 578
values to SQL variables, 985
asterisks
allowed syntax in a CONTAINS clause, 59
allowed syntax in a full text query string, 59
using in subqueries, 725
AT clause
ALTER EVENT statement, 437
CREATE EVENT statement, 573
CREATE EXISTING TABLE statement, 576
CREATE FUNCTION statement, 596
CREATE PROCEDURE statement, 643
CREATE TABLE statement, 692
ATAN function
syntax, 162
ATAN2 function
syntax, 163
ATN2 function
syntax, 163
ATOMIC clause
BEGIN statement, 524
ATTACH TRACING statement
diagnostic tracing, 514
syntax, 514
attachments
email example, 1338
ATTENDED clause
BACKUP statement, 518
attributes
altering remote server using the ALTER SERVER statement, 463
auditing
adding comments, 1093
disabling using sa_disable_auditing_type system procedure, 1132
enabling using sa_enable_auditing_type system procedure, 1134
AUTHENTICATION URL clause
ALTER LDAP SERVER statement, 443
CREATE LDAP SERVER statement, 604
authentication_string option
SET MIRROR OPTION statement, 969
authorities
mapping to compatibility roles, 1269
AUTHORIZATION clause
ALTER SERVICE statement [HTTP web service], 466
ALTER SERVICE statement [SOAP web service], 471
CREATE SERVICE statement [HTTP web service], 468, 664
CREATE SERVICE statement [SOAP web service], 474, 671
AUTO clause
  BACKUP statement, 519
AUTO COMPRESSED clause
  LOAD TABLE statement, 878
AUTO TUNE WRITERS clause
  BACKUP statement, 518
AUTO UPDATE clause
  ALTER STATISTICS statement, 480
auto_add_fan_out option
  SET MIRROR OPTION statement, 969
auto_add_server option
  SET MIRROR OPTION statement, 969
auto_commit option
  Interactive SQL option, 975
auto_failover option
  SET MIRROR OPTION statement, 969
auto_unlock_time option
  ALTER LOGIN POLICY statement, 446
  CREATE LOGIN POLICY statement, 608
AUTOINCREMENT
  @@identity, 85
  GET_IDENTITY function, 250
  resetting the value, 1214
AUTOINCREMENT clause
  CREATE TABLE statement, 690
AUTOINCREMENT default
  CREATE TABLE statement, 489, 695
AutoMultiProgrammingLevel property
  setting with sa_server_option, 1222
AutoMultiProgrammingLevelStatistics property
  setting with sa_server_option, 1222
AUTOSTOP clause
  START DATABASE statement, 993
AvailForOptimization property
  sa_materialized_view_info system procedure, 1176
average function
  AVG function, 164
AVG function
  syntax, 164

back quotes
  database objects, 4
  SQL identifiers, 4
backslashes
  in SQL strings, 8
  not allowed in SQL identifiers, 4
BACKUP statement
  syntax, 516
backup.syb
  about, 518
backups
  BACKUP statement, 516
  creating events using the CREATE EVENT statement, 570
  creating using the BACKUP statement, 516
  restoring databases from, 937
  starting on a read-only database server, 520
  to tape using the BACKUP statement, 516
base 10 logarithm
  LOG10 function, 289
base tables
  CREATE TABLE statement, 701
BASE64_DECODE function
  syntax, 165
BASE64_ENCODE function
  syntax, 166
BEFORE clause
  CONTAINS search condition, 55
  CREATE TRIGGER statement, 713
BEFORE keyword, CONTAINS search condition
  not supported in full text search, 59
BEFORE triggers
  CREATE TRIGGER statement, 713
BEGIN CATCH statement
  syntax, 1022
BEGIN DECLARE statement
  embedded SQL syntax, 728
BEGIN keyword
  TSQL compatibility, 526
BEGIN SNAPSHOT statement
  syntax, 523
BEGIN TRANSACTION statement
  Transact-SQL syntax, 527
BEGIN TRY statement
  syntax, 1022
beginning
  user-defined transactions using the BEGIN TRANSACTION statement, 527
BETWEEN ... AND clause
CREATE EVENT statement, 573
BETWEEN clause
cREATE SPATIAL REFERENCE SYSTEM statement, 673
WINDOW clause, 1053
BETWEEN search condition syntax, 45
BIGINT data type comparing with dates and times, 134 syntax, 97
BIGINT UNSIGNED data type syntax, 97
binary escape characters, 6
BINARY data type syntax, 125
binary data types
BINARY, 125 decoding, 165 encoding, 166 getting from columns, 822 IMAGE, 125 LONG BINARY, 126 UNIQUEIDENTIFIER, 126 VARBINARY, 127
binary files importing, 1334
binary large objects
binary data types, 124 exporting, 1344 GET DATA statement, 822 getting from columns, 822 importing ASE generated BCP files, 880 inserting using the xp_read_file system procedure, 1334 SET statement example, 987 transaction log considerations, 432 binary literals special characters, 6 bind variables describing cursors, 744 EXECUTE statement, 794 OPEN statement, 904 BIND VARIABLES FOR clause DESCRIBE statement [ESQL], 744 BINTOHEX function syntax, 166 bit array conversions about, 139 bit array data types about, 107 LONG VARBIT, 107 VARBIT, 108 bit arrays about, 107 converting, 139 data types, 107 BIT data type syntax, 98 BIT_AND function syntax, 167 BIT_LENGTH function syntax, 168 BIT_OR function syntax, 169 BIT_SUBSTR function syntax, 169 BIT_XOR function syntax, 171 bits converting, 139 bitwise operators syntax, 19 BLANK PADDING clause CREATE DATABASE statement, 550 BLOBs configuring BLOB indexing at table creation, 694 configuring BLOB indexing using ALTER TABLE statement, 492 exporting, 1344 GET DATA statement, 822 getting from columns, 822 importing ASE generated BCP files, 880 INLINE clause, CREATE TABLE statement, 693 inserting using the SET statement, 985 inserting using the xp_read_file system procedure, 1334 PREFIX clause, CREATE TABLE statement, 693 querying within BLOBs, 814 SET statement example, 987 transaction log considerations, 432 BLOCK clause FETCH statement, 802 OPEN statement, 905 block fetches FETCH statement, 802 OPEN statement, 905
blocking
  identifying, 1106
blocks
  identifying, 1106
  troubleshooting, 1168
BOM (byte order mark)
  loading data from UTF-16 or UTF-8 data file, 885
brackets
database objects, 4
  SQL identifiers, 4
branches
  MERGE statement, 897
branching
  MERGE statement, 897
BREAK statement
  Transact-SQL syntax, 529
bulk loading
  LOAD TABLE statement, 873
bulk operations
  unloading data using the UNLOAD statement, 1027
BY LOCK clause
  PREPARE statement [ESQL] statement, 915
BY TIMESTAMP clause
  PREPARE statement [ESQL] statement, 915
BY VALUES clause
  PREPARE statement [ESQL] statement, 915
BYE statement
  Interactive SQL syntax, 797
bypassing optimization
  avoiding using FORCE NO OPTIMIZATION clause, 962
  avoiding using FORCE OPTIMIZATION option, 962
BYTE ORDER MARK
  loading data from UTF-16 or UTF-8 data file, 885
  read or write option in the CSCONVERT function, 198
BYTE ORDER MARK clause
  INPUT statement, 854
  LOAD TABLE statement, 877
  OPENXML operator, 14
  OUTPUT statement, 908
  UNLOAD statement, 1028
BYTE_LENGTH function
  syntax, 171
BYTE_SUBSTR function
  syntax, 172

C
  cache
    flushing, 1140
CACHE clause
  ALTER SEQUENCE statement, 462
  CREATE SEQUENCE statement, 656
CacheSizingStatistics property
  setting with sa_server_option, 1222
calendar date
  ISO 8601, 110
CALIBRATE DBSPACE clause
  ALTER DATABASE statement, 427
CALIBRATE DBSPACE TEMPORARY clause
  ALTER DATABASE statement, 427
CALIBRATE GROUP READ clause
  ALTER DATABASE statement, 427
CALIBRATE PARALLEL READ clause
  ALTER DATABASE statement, 427
CALIBRATE SERVER clause
  ALTER DATABASE statement, 427
calibrating
  cost models using the ALTER DATABASE statement, 426
database servers using the ALTER DATABASE statement, 426
  loading and unloading cost models using
  sa_load_cost_model system procedure, 1167
  loading and unloading cost models using
  sa_unload_cost_model system procedure, 1262
  parallel I/O capabilities, 429
CALL statement
  in Transact-SQL, 796
  syntax, 530
calling procedures
  CALL statement, 530
capabilities
  remote servers, 1349
  SYSCAPABILITY system view, 1349
CAPABILITY clause
  ALTER SERVER statement, 464
CARDINALITY function
  syntax, 173
CASCADE clause
  CREATE TABLE statement, 690
CASE clause
  CREATE DATABASE statement, 551
CASE expression
NULLIF function, 312
    syntax, 24
case sensitivity
    comparison operators, 9
    LIKE search condition, 50
    REGEXP search condition, 52
    REGEXP_SUBSTR function, 330
    SIMILAR TO search condition, 54
CASE statement
    syntax, 532
CaseSensitivity property
    DB_EXTENDED_PROPERTY function, 212
CAST function
    data type conversions, 137
    syntax, 174
catalog
    default system views, 1347
    system tables, 1055
    viewing data in system tables, 1055
catalog data
    viewing, 1055
CATALOG ONLY clause
    RESTORE DATABASE statement, 938
catalog procedures
    alphabetical list, 1092
    Transact-SQL, 1091
catalog procedures (ASE)
    Transact-SQL list, 1089
catalog system procedures
    about, 1085
CatalogCollation property
    DB_EXTENDED_PROPERTY function, 212
CATCH statement
    syntax, 1022
CEILING function
    syntax, 175
CERTIFICATE clause
    CREATE FUNCTION statement [web clients], 590
    CREATE PROCEDURE statement [web clients], 632
certificates
    adding comments using the COMMENT statement, 538
    CREATE CERTIFICATE statement, 547
    dropping certificates, 752
    dropping using the DROP CERTIFICATE statement, 752
information, 1094
    sa_certificate_info system procedure, 1094
    storing, 547
    SYSCERTIFICATE system view, 1350
CHANGE PASSWORD system privilege
    granting using the GRANT statement, 829
    change_password_dual_control option
    ALTER LOGIN POLICY statement, 446
    CREATE LOGIN POLICY statement, 608
CHAR data type
    byte-length semantics, 89
    character-length semantics, 89
    comparing with dates and times, 134
    comparing with NCHAR data type, 132
    syntax, 89
    using DESCRIBE on a CHAR column, 89
CHAR function
    syntax, 176
CHAR_LENGTH function
    syntax, 177
character classes
    sub-character classes, 30
    support special character classes, 30
character data
    storage, 89
    strings, 6
character data types
    about, 89
    CHAR, 89
    LONG NVARCHAR, 91
    LONG VARCHAR, 91
    NCHAR, 92
    NTEXT, 93
    NVARCHAR, 93
    TEXT, 94
    UNIQUEIDENTIFIERSTR, 94
    VARCHAR, 95
    XML, 96
character functions
    alphabetical list, 152
character set conversion
    comparisons between CHAR and NCHAR, 132
    comparisons between data types, 131
    comparisons between numeric data types, 133
    converting NCHAR to CHAR, 138
    lossy conversions, 132
    passwords, CREATE USER statement, 722
    passwords, GRANT CONNECT statement, 833
substitution characters, 132
character set conversion on passwords
ALTER USER statement, 510
character sets
COMPARE function, 179
converting during expression evaluation, 132
lossy character set conversions, 132
SORTKEY function, 364
substitution characters, 132
character strings
about, 6
color substitution
comparisons between CHAR and NCHAR, 132
lossy character set conversions, 132
character-length semantics
CHAR data type, 89
VARCHAR data type, 95
CHARINDEX function
syntax, 177
CharSet property
DB_EXTENDEDPROPERTY function, 212
CharsetConversion option
sa_set_http_option system procedure, 1240
CHECK clause
ALTER DATABASE statement, 428
CREATE DATABASE statement, 551
START DATABASE statement, 994
checksums
altering database settings, 428
creating databases with, 551
starting databases with, 994
VALIDATE CHECKSUM statement, 1045
validating, 1045
child_creation option
SET MIRROR OPTION statement, 969
CLASS clause
ALTER SERVER statement, 464
CREATE SERVER statement, 658
REMOVE JAVA statement, 934
classes
Java methods, 148
removing Java, 934
clause order in SQL
syntax conventions, 422
clauses
order of, SQL conventions, 422
CLEAR statement
Interactive SQL syntax, 537
clearing
Interactive SQL panes, 537
client files
READ_CLIENT_FILE function, 328
WRITE_CLIENT_FILE function, 408
client statement caching
RequestLogFile property, 1222
CLIENTPORT clause
CREATE FUNCTION statement [web clients], 590
CREATE PROCEDURE statement [web clients], 632
Clockwise format
CREATE SPATIAL REFERENCE SYSTEM statement, 679
CLOSE statement
embedded SQL syntax, 537
syntax, 537
close_on_endtrans option
setting with Transact-SQL SET statement, 987
closing
connections using the DROP CONNECTION statement, 753
cursors using the CLOSE statement [ESQL] [SP], 537
Interactive SQL, 797

CLUSTERED clause
ALTER INDEX statement, 441
CREATE INDEX statement, 600

clustered indexes
creating using ALTER INDEX statement, 441

COALESCE function
syntax, 179

code pages
INPUT statement, 855
OUTPUT statement, 908

coefficient of determination
about, 337

COL_LENGTH function
syntax, 154

COL_NAME function
syntax, 154

COLLATION clause
collation tailoring, 552
CREATE DATABASE statement, 552

Collation property
DB_EXTENDED_PROPERTY function, 212
collation sequences
CREATE DATABASE statement, 552
LIKE search condition, 49
collation tailoring
COLLATION clause, CREATE DATABASE statement, 552
COMPARE function, 179
NCHAR COLLATION clause, CREATE DATABASE statement, 554
SORTKEY function, 364
collations
REGEXP search condition, 52
REGEXP_SUBSTR function, 330
SIMILAR TO search condition, 54
SORTKEY function, 364
tailoring at database creation time, 552

CollectStatistics property
setting with sa_server_option, 1222

COLUMN clause
DESCRIBE statement [ESQL], 744
GET DATA statement, 823
column compression
ALTER TABLE statement, 486
CREATE TABLE statement, 690

retrieving compression statistics, 1100
column constraints
adding using the ALTER TABLE statement, 491
changing using the ALTER TABLE statement, 494
column definition
CREATE TABLE statement, 693

COLUMN DELIMITED BY clause
UNLOAD statement, 1028
column names
syntax, 23
column statistics
only partially updated by LOAD TABLE, 886
selectivity estimates, 64
SYSCOLSTAT system view, 1351
SYSCOLSTATS consolidated view, 1426
updating using CREATE STATISTICS, 682

COLUMN WIDTHS clause
INPUT statement, 855
OUTPUT statement, 908
column-name
common element in SQL syntax, 421
columns
aliases, 958
altering using the ALTER TABLE statement, 486
constraints and defaults with domains, 131
constraints in CREATE TABLE statement, 697
domains, 131
getting binary data from, 822
privileges on, 1351
renaming, 496
SQL Remote updating, 1034
SYSTABCOL view, 1401
updating without logging, 1054
user-defined data types, 131
combining
result of multiple SELECT statements, 1025
comma-separated lists
LIST function syntax, 285
commands
executing operating system, 1017

COMMENT statement
syntax, 538
comments
adding to database objects using the COMMENT statement, 538
inserting, 85
removing, 85
syntax, 85
COMMENTS INTRODUCED BY clause
  LOAD TABLE statement, 878
commit
  preparing for two-phase, 917
COMMIT statement
  preparing for two-phase, 917
  referential integrity, 1096
  syntax, 541
committing
  transactions using the COMMIT statement, 541
communication protocols
  multiple settings in MobiLink, 689
COMPARE function
  collation tailoring, 179
  syntax, 179
comparing
  CHAR and NCHAR, 132
  COMPARE function, 179
comparing dates and times
  about, 134
comparison operators
  data conversion, 131
  symbols, 9
  syntax, 9
  Transact-SQL compatibility, 9
comparisons
  CHAR and NCHAR values, 132
  composite data types, 136
  dates, 134
  numeric data types, 133
  search conditions, 40
  times, 134
compatibility
  datatype, 136
  NULLs, 73
  T-SQL expressions and QUOTED IDENTIFIER option, 39
    Transact-SQL comparison operators, 9
  Transact-SQL expressions, 39
  Transact-SQL global variables, 82
  Transact-SQL local variables, 80
  Transact-SQL views, 1449
  views, 1438
compatibility of expressions
  about, 39
compatibility roles
  granting using the GRANT statement, 828
  migrating using ALTER ROLE, 460
  revoking using the REVOKE statement, 945
compatibility views
  about, 1438
  SYSCOLLATION, 1439
  SYSCOLLATIONMAPPINGS, 1439
  SYSCOLUMN, 1439
  SYSFKCOL, 1441
  SYSFOREIGNKEY, 1441
  SYSGROUP, 1442
  SYSGROUPS, 1442
  SYSINDEX, 1443
  SYSINFO, 1443
  SYSEXECOL, 1444
  SYSTABLE, 1444
  SYSTABLEAUTH, 1446
  SYSTABLEAUTHORITY, 1446
  SYSUSERLIST, 1447
  SYUSERPERM, 1447
  SYUSERPERMS, 1448
composite data types
  about, 128
  comparing, 136
compound statements
  BEGIN statement, 523
  BEGIN statement [TSQL], 526
  TRY statement, 1022
  Transact-SQL compatibility, 526
COMPRESS function
  syntax, 181
COMPRESSED clause
  ALTER TABLE statement, 492
  CREATE TABLE statement, 693
  LOAD TABLE statement, 878
  UNLOAD statement, 1028
compressed columns
  ALTER TABLE statement, 486
  retrieving compression statistics, 1100
compressing
  tables using the ALTER TABLE statement, 486
  columns
  CREATE TABLE statement, 693
  compressing strings on Unix
  COMPRESS function, 181
compression
  COMPRESS function, 181
  statistics, 1104
COMPUTE clause
  ALTER TABLE statement, 493
CREATE TABLE statement, 700
COMPUTES clause
LOAD TABLE statement, 878
concatenating strings
array operators, 18
string operators, 11
concurrency
locking tables, 890
condition
common element in SQL syntax, 421
conditions
CONTAINS, 55
EXISTS, 62
search, 40
SQL search conditions, 40
subqueries in, 42
three-valued logic, 63
CONFIGURATION clause
CREATE TEXT INDEX statement, 711
CONFIGURE statement
Interactive SQL syntax, 543
CONFLICT function
syntax, 182
conflicts
CONFLICT function for SQL Remote, 182
CONNECT
revoking using the REVOKE statement, 946
CONNECT privilege
GRANT CONNECT statement, 833
CONNECT statement
embedded SQL syntax, 543
Interactive SQL syntax, 543
CONNECT TO clause
GRANT CONNECT statement, 833
CONNECT TRACING statement
populating sa_diagnostic_tracing_level table, 1081
connecting
creating events using the CREATE EVENT statement, 570
databases using the CONNECT statement, 543
Connection
HTTP_RESPONSE_HEADER function, 268
CONNECTION CLOSE clause
ALTER SERVER statement, 464
CONNECTION RETRIES clause
ALTER LDAP SERVER statement, 444
CREATE LDAP SERVER statement, 604
CONNECTION TIMEOUT clause
ALTER LDAP SERVER statement, 444
CREATE LDAP SERVER statement, 604
connection-level variables
definition, 79
syntax, 81
connection-name
common element in SQL syntax, 421
CONNECTION_EXTENDED_PROPERTY function
syntax, 184
CONNECTION_PROPERTY function
syntax, 186
connection_string option
ALTER MIRROR SERVER statement, 454
CREATE MIRROR SERVER statement, 618
connections
creating events for failed connections, 570
cursor list, 1166
disabling connections to a server, 1222
disallowing with RAISERROR, 920
DROP CONNECTION statement, 753
dropping in Interactive SQL, 751
enabling pooling, 990
generating a list of connection IDs, 1109
setting, 966
setting a maximum number, 920
ConnsDisabled property
setting with sa_server_option, 1222
ConnsDisabledForDB property
setting with sa_server_option, 1222
ConsoleLogFile property
setting with sa_server_option, 1222
ConsoleLogMaxSize property
setting with sa_server_option, 1222
CONSOLIDATE privilege
granting, 834
REVOKE CONSOLIDATE statement, 942
CONSOLIDATED clause
CREATE EVENT statement, 570
consolidated databases
SQL Remote revoking privileges, 942
consolidated views
about, 1419
SYSARTICLECOLS, 1423
SYSARTICLES, 1424
SYSCAPABILITIES, 1424
SYSCATALOG, 1425
SYSCOLAUTH, 1425
SYSCOLSTATS, 1426
about, 152
Cookie
   HTTP_HEADER function, 266
COORDINATE clause
   ALTER SPATIAL REFERENCE SYSTEM statement, 478
   CREATE SPATIAL REFERENCE SYSTEM statement, 677
coordinated universal time
   CURRENT UTC TIMESTAMP, 70
   UTC_TIMESTAMP, 78
COPY clause
   BACKUP statement, 519
copy nodes
   dropping servers, 764
copy servers
   changing with ALTER MIRROR SERVER statement, 453
   defining with CREATE MIRROR SERVER statement, 617
copyright
   retrieving, 1333
CORR function
   syntax, 190
correlation function
   CORR function, 190
correlation names
   DELETE statement, 742
   UPDATE statement, 1039
COS function
   syntax, 191
cosine function
   COS function, 191
cost models
   calibrating the database server, 426
   loading, 1167
   recalibrating using the ALTER DATABASE statement, 426
   unloading, 1262
cost-based optimization
   avoiding using FORCE NO OPTIMIZATION clause, 962
   forcing for procedures, 962
   forcing using FORCE OPTIMIZATION option, 962
COT function
   syntax, 191
cotangent function
   COT function, 191
COUNT clause
   GET_DESCRIPTOR statement [ESQL], 824
   SET_DESCRIPTOR statement [ESQL], 967
COUNT function
   syntax, 192
COUNT_BIG function
   syntax, 193
COUNT_SET_BITS function
   syntax, 195
CounterClockwise format
   CREATE SPATIAL REFERENCE SYSTEM statement, 679
COVAR_POP function
   syntax, 195
COVAR_SAMP function
   syntax, 197
CREATE CERTIFICATE statement
   syntax, 547
CREATE DATABASE statement
   syntax, 549
CREATE DATATYPE statement
   syntax, 562
CREATE DBSPACE statement
   syntax, 557
CREATE DECRYPTED DATABASE statement
   syntax, 558
CREATE DECRYPTED FILE statement
   syntax, 560
CREATE DOMAIN statement
   syntax, 562
   using, 130
CREATE ENCRYPTED DATABASE statement
   syntax, 564
CREATE ENCRYPTED FILE statement
   example of usage to change an encryption key, 569
   syntax, 567
CREATE ENCRYPTED TABLE DATABASE statement
   syntax, 564
CREATE EVENT statement
   syntax, 570
CREATE EXTERNALTABLE statement
   sp_remote_columns system procedure, 1298
   sp_remote_tables system procedure, 1308
   syntax, 576
CREATE EXTERNLOGIN statement
   syntax, 578
CREATE FUNCTION statement
  syntax, 594
  syntax for creating web services functions, 586
  syntax for interfacing to native functions, 580
  Transact-SQL example, 599
CREATE INDEX statement
  syntax, 599
  table use, 602
CREATE LDAP SERVER statement
  syntax, 603
CREATE LOCAL TEMPORARY TABLE statement
  syntax, 606
CREATE LOGIN POLICY statement
  syntax, 607
CREATE MATERIALIZED VIEW statement
  syntax, 612
CREATE MESSAGE statement
  Transact-SQL syntax, 614
CREATE MIRROR SERVER statement
  syntax, 615
CREATE PROCEDURE statement
  sp_remote_pcols system procedure, 1303
  sp_remote_procedures system procedure, 1307
  syntax, 639
  syntax for creating web services procedures, 628
  syntax for interfacing to native procedures, 620
  Transact-SQL syntax, 637
CREATE PUBLICATION statement
  MobiLink syntax, 647
  SQL Remote syntax, 647
CREATE REMOTE [MESSAGE] TYPE statement
  SQL Remote syntax, 651
CREATE ROLE statement
  syntax, 652
CREATE SCHEMA statement
  syntax, 654
CREATE SEQUENCE statement
  syntax, 655
CREATE SERVER statement
  syntax, 657
CREATE SERVICE statement [HTTP web service]
  syntax, 662
CREATE SERVICE statement [SOAP web service]
  syntax, 668
CREATE SPATIAL REFERENCE SYSTEM
  statement
  syntax, 673
CREATE SPATIAL UNIT OF MEASURE statement
  syntax, 681
CREATE STATISTICS statement
  syntax, 682
CREATE SUBSCRIPTION statement
  SQL Remote syntax, 683
CREATE SYNCHRONIZATION PROFILE
  statement
  MobiLink syntax, 685
CREATE SYNCHRONIZATION SUBSCRIPTION
  statement
  MobiLink syntax, 686
CREATE SYNCHRONIZATION USER statement
  MobiLink syntax, 688
CREATE TABLE clause
  INPUT statement, 855
  OUTPUT statement, 908
CREATE TABLE statement
  remote tables, 692
  syntax, 690
  Transact-SQL, 702
CREATE TEMPORARY PROCEDURE statement
  syntax, 639
CREATE TEMPORARY TRACE EVENT SESSION
  statement
  syntax, 707
CREATE TEMPORARY TRACE EVENT statement
  syntax, 705
CREATE TEXT CONFIGURATION statement
  syntax, 709
CREATE TEXT INDEX statement
  syntax, 711
CREATE TRIGGER statement
  syntax, 713
  Transact-SQL syntax, 720
  trigger operation conditions, 716
CREATE USER statement
  character set conversion on passwords, 722
  password maximum length, 722
  passwords, character set conversion on, 722
  syntax, 721
CREATE VARIABLE statement
  syntax, 722
CREATE VIEW statement
  syntax, 724
CREATEDIRS clause
  CREATE SERVER statement, 659
  creating
backups of databases using the BACKUP statement, 516
certificates, 547
CREATE INDEX statement, 599
CREATE MATERIALIZED VIEW statement, 612
CREATE PUBLICATION statement, 647
CREATE SYNCHRONIZATION PROFILE statement, 685
CREATE SYNCHRONIZATION SUBSCRIPTION statement, 686
CREATE TABLE statement, 690
CREATE TRIGGER statement, 713
CREATE VIEW statement, 724
creating users using CREATE USER statement, 721
cursors, 729
database files using the CREATE DBSPACE statement, 557
databases using the CREATE DATABASE statement, 549
LDAP server configuration objects, 603
local temporary tables, 735
local temporary tables using the CREATE LOCAL TEMPORARY TABLE statement, 606
messages, 614
proxy procedures using sp_remote_procedures system procedure, 1307
proxy tables, 692
proxy tables using sp_remote_tables system procedure, 1308
proxy tables using the CREATE EXISTING TABLE statement, 576
savepoints, 954
schemas, 654
servers, 657
SQL Remote remote message types, 651
SQL variables using CREATE VARIABLE statement, 722
SQL variables using DECLARE statement, 736
stored procedures, 639
subscriptions, 683
text configuration objects for full text search, 709
text indexes for full text search, 711
triggers in Transact-SQL, 720
creating databases
CREATE DATABASE statement, 549
creating domains
CREATE DOMAIN statement, 562
creating external logins
CREATE EXTERNLOGIN statement, 578
creating indexes
CREATE INDEX statement, 599
creating login policies
CREATE LOGIN POLICY statement, 607
creating materialized views
CREATE MATERIALIZED VIEW statement, 612
creating synchronization profiles
CREATE SYNCHRONIZATION PROFILE statement [MobiLink], 685
creating synchronization subscriptions
CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 686
creating tables
CREATE TABLE statement, 690
creating views
CREATE VIEW statement, 724
CROSS APPLY clause
FROM clause, 815
CROSS JOIN clause
FROM clause SQL syntax, 810
CSCONVERT function
syntax, 198
CUBE operation
GROUP BY clause, 848
WITH CUBE clause, 848
CUME_DIST function
syntax, 200
CURRENT clause
DISCONNECT statement [ESQL] [Interactive SQL] statement, 751
CURRENT DATABASE clause
CREATE TABLE statement, 690
CURRENT DATABASE special value
syntax, 65
CURRENT DATE clause
CREATE TABLE statement, 690
CURRENT DATE function
TODAY function, 389
CURRENT DATE special value
syntax, 66
CURRENT PUBLISHER clause
CREATE TABLE statement, 690
CURRENT PUBLISHER special value
setting, 839
syntax, 66
CURRENT REMOTE USER clause
CREATE TABLE statement, 690
CURRENT REMOTE USER special value
syntax, 67
CURRENT TIME clause
CREATE TABLE statement, 690
CURRENT TIME special value
syntax, 67
CURRENT_TIMESTAMP clause
CREATE TABLE statement, 690
CURRENT_TIMESTAMP default
CREATE TABLE statement, 489, 694
CURRENT_TIMESTAMP special value
syntax, 68
CURRENT_USER clause
CREATE TABLE statement, 690
CURRENT_USER special value
syntax, 69
CURRENT_UCT_TIMESTAMP clause
CREATE TABLE statement, 690
CURRENT_TIMESTAMP default
CREATE TABLE statement, 489, 695
CURRENT_UCT_TIMESTAMP special value
syntax, 70
CURRENT_TIMESTAMP special value
syntax, 68
CURRENT_USER special value
syntax, 69
CurrentMultiProgrammingLevel property
setting with sa_server_option, 1222
cursors
changing database option settings, 973
CLOSE statement [ESQL] [SP], 537
declaring, 729
declaring in Transact-SQL, DECLARE CURSOR
statement, 732
declaring in Transact-SQL, SELECT statement, 965
deleting rows from, 739
DESCRIBE statement [ESQL], 744
describing behavior in external procedures, 623
describing behavior in user defined procedures, 641
EXPLAIN statement syntax, 799
fetching rows from, 800
inserting rows using, 919
listing per connection, 1166
looping over, 805
OPEN statement, 904
opening, 904
preparing statements, 915
re-describing in external procedures, 623
re-describing in user defined procedures, 641
updatability set in SELECT statement, 960
CYCLE clause
ALTER SEQUENCE statement, 463
CREATE SEQUENCE statement, 656

D
data
exporting from tables into files, 907
importing into tables from files, 854
selecting rows, 955
data access plans
getting text specification, 799
DATA clause
GET DESCRIPTOR statement [ESQL], 824
SET DESCRIPTOR statement [ESQL], 967
data type conversion functions
about, 146
data type conversions
about, 137
CAST, 174
comparing CHAR and NCHAR values, 132
comparison operators, 131
converting DOUBLE to NUMERIC, 140
converting NCHAR to CHAR, 138
Java-to-SQL, 140
SQL to Java, 140
SQL-to-Java, 141
when evaluating expressions, 131
data types
altering using the ALTER DOMAIN statement, 435
BIGINT data type, 97
BINARY data type, 125
BIT data type, 98
CHAR data type, 89
character, 89
comparing values, 131
compatibility, 136
composite data types, 128
converting for comparison operators, 131
converting Java and SQL, 140
CREATE DOMAIN statement, 562
DATE data type, 115
DATETIME data type, 116
DATETIMEOFFSET data type, 117
DECIMAL data type, 99
DOUBLE data type, 100
dropping user-defined using the DROP
dATATYPE statement, 755
FLOAT data type, 101
IMAGE data type, 125
INTEGER data type, 102
LONG BINARY data type, 126
LONG NVARCHAR data type, 91
LONG VARBIT data type, 107
LONG VARCHAR data type, 91
NCHAR, 92
NCHAR (NATIONAL CHAR), 92
NTEXT data type, 93
NUMERIC data type, 103
NVARCHAR data type, 93
REAL data type, 104
retrieving, 246
rounding errors, 97
SMALLDATETIME data type, 119
SMALLINT data type, 105
SMALLMONEY data type, 107
special values, 65
SQL conversion functions, 146
SYSDOMAIN system view, 1356
SYSEXTERNLOGIN system view, 1361
SYSUSERTYPE system view, 1415
TEXT data type, 94
tIME data type, 120
TIMESTAMP data type, 121
TIMESTAMP WITH TIME ZONE data type, 123
TINYINT data type, 105
Unicode, 89
UNIQUEIDENTIFIER data type, 126
UNIQUEIDENTIFIERSTR data type, 94
user-defined domains, 130
VARBINARY data type, 127
VARBINARY data type, 108
VARCHAR data type, 95
XML data type, 96
data-type
common element in SQL syntax, 421
DATABASE clause
CONNECT statement, 543
database cleaner
about, 1098
sa_clean_database system procedure, 1097
data-base encryption
CREATE ENCRYPTED DATABASE statement, 564
data-base extraction
SQL Remote REMOTE RESET statement, 932
data-base files
decrypting using CREATE DECRYPTED FILE
statement, 560
decrypting with CREATE DECRYPTED
DATABASE statement, 558
dropping using the DROP DATABASE statement, 754
encrypting using the CREATE ENCRYPTED
DATABASE statement, 564
encrypting using the CREATE ENCRYPTED
FILE statement, 567
storing indexes in, 601
data-base ID numbers
DB_ID function, 216
data-base mirroring
ALTER MIRROR SERVER statement, 452
CREATE MIRROR SERVER statement, 615
dropping servers, 764
initiating a failover, 428
LOAD TABLE statement restrictions, 886
SET MIRROR OPTION statement, 968
data-base names
returning with the DB_NAME function, 217
data-base objects
adding comments using the COMMENT statement, 538
identifying, 4
data-base options
fetching rows from cursors, 973
initial settings and sp_login_environment system
procedure, 1289
initial settings and sp_tsql_environment system
procedure, 1320
quoted_identifier and T-SQL compatibility, 39
setting in Transact-SQL, 987
Transact-SQL compatibility, 1320
user-defined options, 972
data-base schemas
system tables, 1055
system views, 1347
data-base server messages window
displaying messages on, 899
database servers
  database mirroring, ALTER MIRROR SERVER
  statement, 452
  database mirroring, CREATE MIRROR SERVER
  statement, 615
  read-only scale-out, ALTER MIRROR SERVER
  statement, 452
  read-only scale-out, CREATE MIRROR SERVER
  statement, 615
  setting options with sa_server_option system
  procedure, 1222
  START SERVER statement, 996
  STOP SERVER statement, 1007
DATABASE SIZE clause
  CREATE DATABASE statement, 552
  DATABASE SIZE clause
  CREATE DATABASE statement, 552
database validation
  VALIDATE CHECKSUM statement, 1045
  VALIDATE INDEX statement, 1045
DatabaseCleaner property
  setting with sa_server_option, 1222
databases
  backing up using the BACKUP statement, 516
  checkpointing using the CHECKPOINT statement, 536
  connecting to using the CONNECT statement, 543
  creating an encrypted copy of a database, 564
  creating files using the CREATE DBSPACE
  statement, 557
  creating using the CREATE DATABASE
  statement, 549
default system views, 1347
disabling connections, 1222
dropping files using the DROP DATABASE
  statement, 754
loading bulk data into, 873
migrating, 1183
restoring from archives, 937
returning the location of the current database, 215
schema (list of tables), 1347
schema (list of views), 1055
starting, 993
SYSFILE system view, 1440
system procedures, 1085
system tables, 1055
unloading data using the UNLOAD statement, 1027
upgrading jConnect using the ALTER
  DATABASE statement, 426
validating using sa_validate system procedure,
  1266
DataLastModified property
  sa_materialized_view_info system procedure, 1176
DATEDATALENGTH function
  syntax, 201
DataStatus property
  sa_materialized_view_info system procedure, 1176
DATATYPE clause
  ALTER SERVICE statement [SOAP web service],
  471
  CREATE SERVICE statement [SOAP web
  service], 473, 669
Date
  HTTP_RESPONSE_HEADER function, 268
date and time data types
  about, 108
  DATE, 115
  date formats, 110
date first option
  with sa_server_option system
  procedure, 1222
date formats, 110
  DATE, 115
  date with time formats, 113
  DATETIME, 116
  DATETIMEOFFSET, 117
  ISO 8601, 109
  SMALLDATETIME, 119
date formats, 112
time formats, 112
  TIME, 120
time zone formats, 113
timezone formats, 113
  TIMESTAMP, 121
  TIMESTAMP WITH TIME ZONE, 123
date data type
  comparing with dates and times, 134
  ISO 8601 calendar date, 110
  ISO 8601 ordinal date, 110
  ISO 8601 week date, 110
  resolving ambiguous dates, 111
  sending dates and times to the database, 109
  syntax, 115
DATE function
  syntax, 203
date functions
  alphabetical list, 146
date functions
  syntax, 204
DATEDIFF function
  syntax, 205
date functions
  syntax, 205
date first option
Index

SET statement syntax, 987
DATEFORMAT function
syntax, 206
DATENAME function
syntax, 207
DATEPART function
syntax, 208
dates
comparing, 134
conversion functions, 146
converting from strings, 136
February 29, 115
generating table of, 1217
inserting, 115
interpretation, 115
leap years, 115
queries, 114
query the current system date, 252
retrieving, 115
sending to the database, 109
SQL Anywhere, 108
storing, 109
datetime
conversion functions, 146
DATETIME data type
sending dates and times to the database, 109
syntax, 116
DATETIME function
syntax, 209
DATETIMEOFFSET data type
sending dates and times to the database, 109
syntax, 117
DAY function
syntax, 210
day of week
DOW function, 225
DAYNAME function
syntax, 210
DAYS function
syntax, 211
DB2
migrating to SQL Anywhere using sa_migrate system procedure, 1184
db_charset
CSCONVERT function, 198
DB_EXTENDED_PROPERTY function
syntax, 212
DB_ID function
syntax, 216
DB_NAME function
syntax, 217
DB_PROPERTY function
syntax, 218
db_publisher option
setting using GRANT PUBLISH statement, 839
db_register_a_callback function
using with MESSAGE TO CLIENT, 902
DBA PASSWORD clause
CREATE DATABASE statement, 552
DBA USER clause
CREATE DATABASE statement, 552
DBFILE ONLY clause
BACKUP statement, 517
DBFreePercent event condition
about, 238
DBFreeSpace event condition
about, 238
dblsync utility
specifying location using ALTER EXTERNAL ENVIRONMENT statement, 438
dbname FORCE START clause
ALTER DATABASE statement, 428
dbo system role
granting using the GRANT statement, 828
revoking using the REVOKE statement, 945
dbo user
RowGenerator system table, 1084
Transact-SQL compatibility views, 1449
DBSize event condition
about, 238
dbspaces
adding comments using the COMMENT statement, 538
adding pages, 433
altering using the ALTER DBSPACE statement, 432
creating using the CREATE DBSPACE statement, 557
decrypting using CREATE DECRYPTED DATABASE statement, 558
decrypting using CREATE DECRYPTED FILE statement, 560
determining available space, 1133
dropping using the DROP DBSPACE statement, 755
encrypting using the CREATE ENCRYPTED DATABASE statement, 564
encrypting using the CREATE ENCRYPTED FILE statement, 567
granting usage privileges, 836
SYSFILE system view, 1440
deadlock reporting
sa_report_deadlocks system procedure, 1212
deadlock Logging property
setting with sa_server_option, 1222
DeadlockLogging property
setting with sa_server_option, 1222
deadlocks
logging, 1222
sa_report_deadlocks system procedure, 1212
DEALLOCATE DESCRIPTOR statement
embedded SQL syntax, 727
DEALLOCATE statement
syntax, 728
deallocating
descriptor areas, 727
DEBUG ONLY clause
MESSAGE statement, 900
debugging
controlling MESSAGE statement behavior, 899
TRACEBACK, 389
DebuggingInformation property
setting with sa_server_option, 1222
DECIMAL data type
syntax, 99
DECLARE CURSOR statement
embedded SQL syntax, 729
syntax, 729
Transact-SQL syntax, 732
DECLARE EXCEPTION
used with BEGIN statement, 523
used with TRY statement, 1022
DECLARE LOCAL TEMPORARY TABLE statement
syntax, 735
DECLARE statement
syntax, 736
used with BEGIN statement, 523
used with BEGIN statement [TSQL], 526
used with TRY statement, 1022
declaring
cursors, 729
cursors in Transact-SQL, DECLARE CURSOR statement, 732
cursors in Transact-SQL, SELECT statement, 965
exceptions using DECLARE statement, 736
host variables in embedded SQL, 728
variables using DECLARE statement, 736
decoding data
BASE64_DECODE function, 165
HTML_DECODE function, 261
HTTP_DECODE function, 264
DECOMPRESS function
syntax, 219
DECRYPT function
syntax, 220
decrypting
files using CREATE DECRYPTED DATABASE statement, 558
files using CREATE DECRYPTED FILE statement, 560
materialized views using SQL, 450
tables using the ALTER TABLE statement, 486
DEFAULT
specifying in a CALL statement, 530
DEFAULT clause
ALTER TABLE statement, 488
CREATE DOMAIN statement, 562
CREATE SERVICE statement [HTTP web service], 662
CREATE SERVICE statement [SOAP web service], 668
CREATE TABLE statement, 694
special values, 489, 694
DEFAULT LAST USER
avoid replicating columns in SQL Remote, 863
default login policy
about, 607
DEFAULT TIMESTAMP columns
TIMESTAMP special value, 76
default values
CURRENT DATABASE, 65
CURRENT DATE, 66
CURRENT PUBLISHER, 66
CURRENT REMOTE USER, 67
CURRENT TIME, 67
CURRENT TIMESTAMP, 68
CURRENT USER, 69
CURRENT UTC TIMESTAMP, 70
CURRENT_TIMESTAMP, 68
| CURRENT_USER, 69 | ALTER SYNCHRONIZATION USER statement [MobiLink], 485 |
| INSERT statement, 861 | DELETE privilege |
| LAST USER, 71 | GRANT statement, 828 |
| SQLCODE, 74 | REVOKE statement, 946 |
| SQLSTATE, 74 | DELETE privilege clause |
| TIMESTAMP, 76 | GRANT statement, 830 |
| USER, 77 | DELETE statement |
| using DEFAULT for optional parameters, 530 | (positioned) statement syntax, 739 |
| UTC TIMESTAMP, 78 | setting database options, 743 |
| DEFAULT VALUES clause | syntax, 741 |
| INSERT statement, 861 | DELETE statement (positioned) |
| defaults | embedded sql syntax, 739 |
| CREATE TABLE statement, 694 | DELETE TABLE clause |
| INSERT statement, 861 | ALTER PUBLICATION statement, 457 |
| using in domains, 564 | DELETE TYPE clause |
| DEFAULTS clause | ALTER EVENT statement, 437 |
| LOAD TABLE statement, 878 | deleting |
| DEFINITION clause | all rows from a table, 1019 |
| ALTER SPATIAL REFERENCE SYSTEM statement, 478 | columns using the ALTER TABLE statement, 486 |
| CREATE SPATIAL REFERENCE SYSTEM statement, 675 | database files using the DROP DATABASE statement, 754 |
| definitions | dbspaces using the DROP DBSPACE statement, 755 |
| altering tables using the ALTER TABLE statement, 486 | domains, 756 |
| defragmenting | events using the DROP EVENT statement, 757 |
| REORGANIZE TABLE, 935 | functions using the DROP FUNCTION statement, 758 |
| DEGREES function | granting privileges, 944 |
| syntax, 222 | indexes using the DROP INDEX statement, 759 |
| DELAY clause | Java classes, 934 |
| WAITFOR statement, 1048 | materialized views using the DROP |
| DELETE ALL OPTION clause | MATERIALIZED VIEW statement, 762 |
| ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 483 | optimizer statistics using the DROP STATISTICS |
| ALTER SYNCHRONIZATION USER statement [MobiLink], 485 | statement, 775 |
| DELETE clause | prepared statements using the DROP |
| CREATE LOCAL TEMPORARY TABLE statement, 606 | STATEMENT statement, 775 |
| CREATE PUBLICATION statement [MobiLink] [SQL Remote], 647 | procedures using the DROP PROCEDURE |
| CREATE TRIGGER statement, 714 | statement, 765 |
| DECLARE LOCAL TEMPORARY TABLE statement, 735 | rows from cursors, 739 |
| MERGE statement, 896 | rows from databases, 741 |
| SQL variables using the DROP VARIABLE statement, 787 | CREATE TRIGGER statement, 714 |
| START SYNCHRONIZATION DELETE statement, 1001 | CREATE TRIGGER statement, 714 |
| STOP SYNCHRONIZATION DELETE statement, 1010 | CREATE TRIGGER statement, 714 |
| tables using the DROP TABLE statement, 780 | CREATE TRIGGER statement, 714 |
text configuration objects, 781
triggers using the DROP FUNCTION statement, 758
triggers using the DROP statement, 785
views using the DROP VIEW statement, 788
DELIMITED BY clause
LOAD TABLE statement, 878
OUTPUT statement, 908
UNLOAD statement, 1028
DELIMITED clause
INPUT statement, 855
delimited strings
compatibility with ASE, 39
delimiting SQL strings
about, 4
DENSE_RANK function
syntax, 222
denying
granting privileges, 944
dependencies
determining using sa_dependent_views system procedure, 1123
dependent variables
regression line, 332
derived tables
example in a FROM clause, 813
FROM clause SQL syntax, 810
lateral, 813
using in subqueries, 725
DESCRIBE clause
PREPARE statement, 915
DESCRIBE CONNECTION statement
Interactive SQL syntax, 747
DESCRIBE statement
DT_DESCRIBE_INPUT, 745
DT_HIDDEN_COLUMN, 730
DT_KEY_COLUMN, 730
DT_PROCEDURE_IN, 745
DT_PROCEDURE_OUT, 745
embedded SQL syntax, 744
Interactive SQL syntax, 747
long column names, 745
describing
cursor behavior in external procedures, 623
cursor behavior in user defined procedures, 641
cursors, 744
descriptor areas
allocating memory for, 424
deallocating, 727
EXECUTE statement, 794
getting information from, 824
setting, 967
UPDATE (positioned) statement, 1032
DESCRIPTOR clause
EXECUTE statement, 794
descriptors
DESCRIBE statement [ESQL], 744
FETCH statement, 800
preparing statements, 915
DETACH TRACING statement
diagnostic tracing, 750
syntax, 750
deterministic behavior
external functions, 581
user-defined functions, 595
diagnostic tracing
ATTACH TRACING statement, 514
deleting records, 1211
DETACH TRACING statement, 750
REFRESH TRACING LEVEL statement, 930
sa_diagnostic_auxiliary_catalog table, 1070
sa_diagnostic_blocking table, 1070
sa_diagnostic_cachecontents table, 1071
sa_diagnostic_connection table, 1072
sa_diagnostic_cursor table, 1073
sa_diagnostic_deadlock table, 1074
sa_diagnostic_hostvariable table, 1075
sa_diagnostic_internalvariable table, 1076
sa_diagnostic_query table, 1076
sa_diagnostic_request table, 1078
sa_diagnostic_statement table, 1079
sa_diagnostic_statistics table, 1080
sa_diagnostic_tracing_level table, 1081
sa_save_trace_data system procedure, 1217
sa_set_tracing_level system procedure, 1245
tables, about, 1069
diagnostic tracing level
setting at the command line, 1245
diagnostics
sa_performance_statistics system procedure, 1202
DIAGNOSTICS system role
granting using the GRANT statement, 828
revoking using the REVOKE statement, 945
DIFFERENCE function
syntax, 224
directory access servers
Index

CREATE SERVER statement, 657
DIRECTORY clause
   BACKUP statement, 517
   START DATABASE statement, 994
DISABLE clause
   ALTER EVENT statement, 436
   ALTER MATERIALIZED VIEW statement, 450
   ALTER SERVICE statement [HTTP web service], 466
   ALTER SERVICE statement [SOAP web service], 471
   ALTER VIEW statement, 512
   CREATE SERVICE statement [HTTP web service], 468, 665
   CREATE SERVICE statement [SOAP web service], 474, 671
DISABLE USE IN OPTIMIZATION clause
   ALTER MATERIALIZED VIEW statement, 450
DISABLE VIEW DEPENDENCIES clause
   ALTER TABLE statement, 497
disabling connections
   to all databases on a server, 1222
   to individual databases, 1222
DISCONNECT statement
   embedded SQL syntax, 751
   Interactive SQL syntax, 751
disconnecting
   creating events using the CREATE EVENT statement, 570
   DROP CONNECTION statement, 753
DISH
   CREATE SERVICE statement [SOAP web service], 472, 669
DISH services
   forward slashes not allowed in name, 472, 669
disk sandbox
   starting databases with, 995
disk space
   creating events using the CREATE EVENT statement, 570
   creating out of disk space events, 570
determining available space, 1133
disk transfer time model
   calibrating using the ALTER DATABASE statement, 426
current value, 1143
   restoring the default using the ALTER DATABASE statement, 426
DISKSANDBOX clause
   START DATABASE statement, 995
DiskSandbox property
   setting with sa_db_option, 1121
   setting with sa_server_option, 1222
displaying
   messages, 899
   messages in the messages window, 918
DISTINCT clause
   MEDIAN function, 293
   NULL, 72
   SELECT statement, 957
DISTINCT keyword
   about, 957
Distinguished Name (DN)
   ALTER LDAP SERVER statement, 442
   ALTER LOGIN POLICY statement, 445
   ALTER USER statement, 509
   CREATE LDAP SERVER statement, 603
   sa_get_user_status system procedure, 1152
   SYSLDAPSERVER system view, 1369
   SYSUSER system view, 1412
DML
   selecting from, 815
DO clause
   FOR statement, 805
documentation
   conventions for SQL syntax, 421
DOMAIN | DATATYPE clause
   CREATE DOMAIN statement, 562
domains
   about, 130
   altering using the ALTER DOMAIN statement, 435
   CREATE DOMAIN statement, 562
   dropping using the DROP DOMAIN statement, 756
   nullability, 562
   Transact-SQL, 131
DOUBLE data type
   comparing with dates and times, 134
   converting to NUMERIC, 140
   syntax, 100
double hyphen
   comment indicator, 86
double quotes
   database objects, 4
   not allowed in SQL identifiers, 4
double slash
    comment indicator, 86
DOW function
    syntax, 225
download-only
    CREATE PUBLICATION syntax, 647
DRIVER clause
    INPUT statement, 854
DriveType property
    DB_EXTENDED_PROPERTY function, 212
DROP CERTIFICATE statement
    syntax, 752
DROP clause
    ALTER TABLE statement, 496
    DROP EXTERNLOGIN statement, 758
DROP CONNECTION statement
    syntax, 753
DROP DATABASE statement
    syntax, 754
DROP DATATYPE statement
    syntax, 755
DROP DBSPACE statement
    syntax, 755
DROP DOMAIN statement
    syntax, 756
DROP EVENT statement
    syntax, 757
DROP EXTERNLOGIN statement
    syntax, 758
DROP FUNCTION statement
    syntax, 758
DROP INDEX statement
    syntax, 759
DROP LDAP SERVER statement
    syntax, 760
DROP LOGIN POLICY statement
    syntax, 761
DROP MATERIALIZED VIEW statement
    syntax, 762
DROP MESSAGE statement
    syntax, 763
DROP MIRROR SERVER statement
    syntax, 764
DROP PCTFREE clause
    ALTER MATERIALIZED VIEW statement, 451
DROP PREFILTER clause
    ALTER TEXT CONFIGURATION statement, 502
DROP PROCEDURE statement
    syntax, 765
DROP PUBLICATION statement
    MobiLink syntax, 766
    SQL Remote syntax, 766
DROP REMOTE CONNECTION statement
    syntax, 768
DROP REMOTE MESSAGE TYPE statement
    SQL Remote syntax, 767
DROP ROLE statement
    syntax, 769
DROP SEQUENCE statement
    syntax, 771
DROP SERVER statement
    syntax, 772
DROP SERVICE statement
    syntax, 772
DROP SPATIAL REFERENCE SYSTEM statement
    syntax, 773
DROP SPATIAL UNIT OF MEASURE statement
    syntax, 774
DROP STATEMENT statement
    embedded SQL syntax, 775
DROP STATISTICS statement
    syntax, 775
DROP STOPLIST clause
    ALTER TEXT CONFIGURATION statement, 501
DROP SUBSCRIPTION statement
    SQL Remote syntax, 776
DROP SYNCHRONIZATION PROFILE statement
    MobiLink syntax, 778
DROP SYNCHRONIZATION SUBSCRIPTION statement
    syntax, 778
DROP SYNCHRONIZATION USER statement
    MobiLink syntax, 779
DROP TABLE clause
    ALTER PUBLICATION statement, 457
    ALTER TABLE statement, 780
DROP TABLE statement
    syntax, 780
DROP TEXT CONFIGURATION statement
    syntax, 781
DROP TEXT INDEX statement
    syntax, 782
DROP TRACE EVENT SESSION statement
    syntax, 784
DROP TRACE EVENT statement
    syntax, 783
DROP TRIGGER statement
syntax, 785
DROP USER statement
    syntax, 786
DROP VARIABLE statement
    syntax, 787
DROP VIEW statement
    syntax, 788
DropBadStatistics property
    setting with sa_server_option, 1222
dropping
    certificates using the DROP CERTIFICATE statement, 752
columns using the ALTER TABLE statement, 486
collections in Interactive SQL, 751
collections using the DROP COLLECTION statement, 753
database files using the DROP DATABASE statement, 754
dbspaces using the DROP DBSPACE statement, 755
domains using the DROP DOMAIN statement, 756
DROP PUBLICATION statement, 766
DROP SUBSCRIPTION statement, 776
DROP SYNCHRONIZATION SUBSCRIPTION statement, 778
DROP SYNCHRONIZATION USER statement, 779
events using the DROP EVENT statement, 757
functions using the DROP FUNCTION statement, 758
indexes using the DROP INDEX statement, 759
LDAP server configuration objects, 760
login policies using DROP LOGIN POLICY statement, 761
login policies using DROP USER statement, 786
logins for remote servers, 758
materialized views using the DROP MATERIALIZED VIEW statement, 762
messages using the DROP MESSAGE statement, 763
optimizer statistics using the DROP STATISTICS statement, 775
prepared statements using the DROP STATEMENT statement, 775
procedures using the DROP PROCEDURE statement, 765
remote message types, 767
remote servers using the DROP SERVER statement, 772
roles using the DROP ROLE statement, 769
spatial reference system using the DROP SPATIAL REFERENCE SYSTEM statement, 773
SQL variables using the DROP VARIABLE statement, 787
tables using the DROP TABLE statement, 780
text configuration objects, 781
text indexes for full text search, 782
triggers using the DROP FUNCTION statement, 758
triggers using the DROP statement, 785
users using the REVOKE statement, 944
views using the DROP VIEW statement, 788
web services using the DROP SERVICE statement, 772
dropping connections
    DROP CONNECTION statement, 753
dropping indexes
    DROP statement, 759
dropping materialized views
    DROP MATERIALIZED VIEW statement, 762
dropping remote procedures
    DROP PROCEDURE statement, 765
dropping synchronization profiles
    DROP SYNCHRONIZATION PROFILE statement [MobiLink], 778
dropping tables
    DROP TABLE statement, 780
dropping views
    DROP VIEW statement, 788
DropUnusedStatistics property
    setting with sa_server_option, 1222
DSN clause
    INPUT statement, 854
DT_DESCRIBE_INPUT
    DESCRIBE statement, 745
DT_HIDDEN_COLUMN
    DESCRIBE statement, 730
DT_KEY_COLUMN
    DESCRIBE statement, 730
DT_PROCEDURE_IN
    DESCRIBE statement, 745
DT_PROCEDURE_OUT
    DESCRIBE statement, 745
dual control passwords
    sa_get_user_status system procedure, 1152
DUMMY
   Row Constructor algorithm, 1055
   system table, 1055
DYNAMIC RESULT SETS clause
   CREATE PROCEDURE statement [External call], 623
DYNAMIC SCROLL clause
   DECLARE CURSOR statement, 730
   FOR statement, 805
DYNAMIC SCROLL cursors
   declaring, 729
dynamic SQL
   executing procedures in, 791

E
   elements
      SQL language syntax, 1
ELLIPSOID clause
   ALTER SPATIAL REFERENCE SYSTEM
      statement, 479
   CREATE SPATIAL REFERENCE SYSTEM
      statement, 678
ELSE
   CASE expression, 24
   IF expressions, 23
ELSE clause
   IF statement, 850
   IF statement [T-SQL], 852
ELSEIF clause
   IF statement, 850
email
   extended system procedures, 1086
   system procedures, 1086
   xp_sendmail system procedure, 1336
embedded SQL
   ALLOCATE DESCRIPTOR syntax, 424
   BEGIN DECLARE statement syntax, 728
   CLOSE statement syntax, 537
   CONNECT statement syntax, 543
   DEALLOCATE DESCRIPTOR statement syntax, 727
   DECLARE CURSOR statement syntax, 729
   DELETE (positioned) statement syntax, 739
   DESCRIBE statement syntax, 744
   DISCONNECT statement syntax, 751
   DROP STATEMENT statement syntax, 775
   END DECLARE statement syntax, 728
   EXECUTE IMMEDIATE statement syntax, 791
   EXECUTE statement syntax, 794
   EXPLAIN statement syntax, 799
   FETCH statement syntax, 800
   GET DATA statement syntax, 822
   GET DESCRIPTOR statement syntax, 824
   GET OPTION statement syntax, 825
   INCLUDE statement syntax, 853
   OPEN statement syntax, 904
   PREPARE statement syntax, 915
   PUT statement syntax, 919
   SET CONNECTION statement syntax, 966
   SET DESCRIPTOR statement syntax, 967
   SET SQLCA statement syntax, 984
   UPDATE (positioned) statement, 1032
   WHENEVER statement syntax, 1049
ENABLE clause
   ALTER MATERIALIZED VIEW statement, 450
   ALTER SERVICE statement [HTTP web service], 466
   ALTER SERVICE statement [SOAP web service], 471
   ALTER VIEW statement, 512
   CREATE SERVICE statement [HTTP web
      service], 468, 665
   CREATE SERVICE statement [SOAP web
      service], 474, 671
ENABLE USE IN OPTIMIZATION clause
   ALTER MATERIALIZED VIEW statement, 450
ENABLE | DISABLE clause
   CREATE EVENT statement, 573
encoding
   INPUT statement, 855
   LOAD TABLE syntax, 873
   OUTPUT statement, 908
   READ statement, 922
   UNLOAD statement, 1027
ENCODING clause
   CREATE DATABASE statement, 552
   INPUT statement, 855
   LOAD TABLE statement, 879
   OUTPUT statement, 908
   READ statement [Interactive SQL], 922
   UNLOAD statement, 1028
encoding data
   BASE64_ENCODE function, 166
   HTML_ENCODE function, 262
encodings
<table>
<thead>
<tr>
<th>CREATE DATABASE statement, 552</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCRYPT function</td>
</tr>
<tr>
<td>syntax, 226</td>
</tr>
<tr>
<td>ENCRYPTED clause</td>
</tr>
<tr>
<td>ALTER MATERIALIZED VIEW statement, 451</td>
</tr>
<tr>
<td>CREATE DATABASE statement, 553</td>
</tr>
<tr>
<td>CREATE TABLE statement, 692</td>
</tr>
<tr>
<td>LOAD TABLE statement, 879</td>
</tr>
<tr>
<td>UNLOAD statement, 1028</td>
</tr>
<tr>
<td>ENCRYPTED DATABASE clause</td>
</tr>
<tr>
<td>CREATE ENCRYPTED DATABASE statement, 565</td>
</tr>
<tr>
<td>ENCRYPTED TABLE clause</td>
</tr>
<tr>
<td>CREATE DATABASE statement, 553</td>
</tr>
<tr>
<td>ENCRYPTED TABLE DATABASE clause</td>
</tr>
<tr>
<td>CREATE ENCRYPTED TABLE DATABASE statement, 565</td>
</tr>
<tr>
<td>encrypting</td>
</tr>
<tr>
<td>databases, CREATE ENCRYPTED DATABASE statement, 564</td>
</tr>
<tr>
<td>materialized views using SQL, 450</td>
</tr>
<tr>
<td>encrypting tables</td>
</tr>
<tr>
<td>ALTER TABLE statement, 486</td>
</tr>
<tr>
<td>encryption</td>
</tr>
<tr>
<td>CREATE DATABASE statement, 553</td>
</tr>
<tr>
<td>CREATE DECRYPTED DATABASE statement, 558</td>
</tr>
<tr>
<td>CREATE ENCRYPTED FILE statement, 567</td>
</tr>
<tr>
<td>encryption algorithms</td>
</tr>
<tr>
<td>CREATE DATABASE statement, 553</td>
</tr>
<tr>
<td>encryption keys</td>
</tr>
<tr>
<td>changing the key for an encrypted database, 569</td>
</tr>
<tr>
<td>END</td>
</tr>
<tr>
<td>CASE expression, 24</td>
</tr>
<tr>
<td>END CASE</td>
</tr>
<tr>
<td>CASE expression, 24</td>
</tr>
<tr>
<td>END CATCH statement</td>
</tr>
<tr>
<td>syntax, 1022</td>
</tr>
<tr>
<td>END DECLARE statement</td>
</tr>
<tr>
<td>embedded SQL syntax, 728</td>
</tr>
<tr>
<td>END FOR clause</td>
</tr>
<tr>
<td>FOR statement, 805</td>
</tr>
<tr>
<td>END IF</td>
</tr>
<tr>
<td>IF expressions, 23</td>
</tr>
<tr>
<td>END IF clause</td>
</tr>
<tr>
<td>IF statement, 850</td>
</tr>
<tr>
<td>END keyword</td>
</tr>
<tr>
<td>TSQL compatibility, 526</td>
</tr>
<tr>
<td>END LOOP clause</td>
</tr>
<tr>
<td>LOOP statement, 891</td>
</tr>
<tr>
<td>END statement</td>
</tr>
<tr>
<td>used with BEGIN statement, 523</td>
</tr>
<tr>
<td>used with BEGIN statement [TSQL], 526</td>
</tr>
<tr>
<td>END TRY statement</td>
</tr>
<tr>
<td>syntax, 1022</td>
</tr>
<tr>
<td>ENDIF</td>
</tr>
<tr>
<td>IF expressions, 23</td>
</tr>
<tr>
<td>ENDIF clause</td>
</tr>
<tr>
<td>IF statement, 850</td>
</tr>
<tr>
<td>ending</td>
</tr>
<tr>
<td>rolling back transactions, 950</td>
</tr>
<tr>
<td>engines</td>
</tr>
<tr>
<td>starting database, 996</td>
</tr>
<tr>
<td>stopping database, 1007</td>
</tr>
<tr>
<td>ENVIRONMENT clause</td>
</tr>
<tr>
<td>INSTALL EXTERNAL OBJECT statement, 866</td>
</tr>
<tr>
<td>environment variables</td>
</tr>
<tr>
<td>getting, 1332</td>
</tr>
<tr>
<td>xp_getenv system procedure, 1332</td>
</tr>
<tr>
<td>environments</td>
</tr>
<tr>
<td>altering using ALTER EXTERNAL ENVIRONMENT statement, 438</td>
</tr>
<tr>
<td>equal to</td>
</tr>
<tr>
<td>comparison operator, 9</td>
</tr>
<tr>
<td>error messages</td>
</tr>
<tr>
<td>ERRORMSG function, 235</td>
</tr>
<tr>
<td>ERROR_LINE function</td>
</tr>
<tr>
<td>syntax, 228</td>
</tr>
<tr>
<td>ERROR_MESSAGE function</td>
</tr>
<tr>
<td>syntax, 229</td>
</tr>
<tr>
<td>ERROR_PROCEDURE function</td>
</tr>
<tr>
<td>syntax, 230</td>
</tr>
<tr>
<td>ERROR_SQLCODE function</td>
</tr>
<tr>
<td>syntax, 231</td>
</tr>
<tr>
<td>ERROR_SQLSTATE function</td>
</tr>
<tr>
<td>syntax, 232</td>
</tr>
<tr>
<td>ERROR_STACK_TRACE function</td>
</tr>
<tr>
<td>syntax, 233</td>
</tr>
<tr>
<td>ERRORMSG function</td>
</tr>
<tr>
<td>syntax, 235</td>
</tr>
<tr>
<td>ErrorNumber event condition</td>
</tr>
<tr>
<td>about, 238</td>
</tr>
<tr>
<td>errors</td>
</tr>
<tr>
<td>creating events using the CREATE EVENT statement, 570</td>
</tr>
<tr>
<td>RAISERROR statement, 920</td>
</tr>
</tbody>
</table>
signaling, 992
trapping in embedded SQL, 1049
user-defined messages, 1414
ESCAPE CHARACTER clause
  INPUT statement, 856
  LOAD TABLE statement, 879
  OUTPUT statement, 908
  UNLOAD statement, 1029
escape characters
  about , 8
  binary literals, 6
  creating databases, 549
  INPUT statement, 854
  OUTPUT statement, 907
ESCAPE clause
  LIKE search condition, 47
escape sequences
  backslashes in SQL strings, 8
  hexadecimal values in SQL strings, 8
  new line characters in SQL strings, 8
  single quotes in SQL strings, 8
  Unicode, 398
  Unicode values in SQL strings, 8
ESCAPES clause
  INPUT statement, 856
  LOAD TABLE statement, 880
  OUTPUT statement, 909
  UNLOAD statement, 1029
ESQL
  statement indicators, 424
establishing
  savepoints, 954
ESTIMATE function
  syntax, 236
ESTIMATE_SOURCE function
  syntax, 237
estimates
  explicit selectivity estimates, 64
EvenOdd format
  CREATE SPATIAL REFERENCE SYSTEM
  statement, 679
event conditions
  list, 238
event handlers
  hiding using the ALTER EVENT statement, 436
event tracing
  ALTER TRACE EVENT SESSION statement, 506
  CREATE TEMPORARY TRACE EVENT
  SESSION statement, 707
  CREATE TEMPORARY TRACE EVENT
  statement, 705
  DROP TRACE EVENT SESSION statement, 784
  DROP TRACE EVENT statement, 783
  NOTIFY TRACE EVENT statement, 903
EVENT_CONDITION function
  syntax, 238
EVENT_CONDITION_NAME function
  syntax, 240
EVENT_PARAMETER function
  syntax, 240
events
  adding comments using the COMMENT statement, 538
  ALTER EVENT statement, 436
  altering using the ALTER EVENT statement, 436
  CREATE EVENT statement, 570
  disabling, 436
  DROP EVENT statement, 757
  dropping using the DROP EVENT statement, 757
  EVENT_PARAMETER, 240
  scheduling using the ALTER EVENT statement, 436
  scheduling using the CREATE EVENT statement, 570
  TRIGGER EVENT statement, 1018
  triggering, 1018
EVERY clause
  CREATE EVENT statement, 573
EXCEPT clause
  setting database options, 790
EXCEPT statement
  syntax, 789
EXCEPTION clause
  BEGIN statement, 523
exceptions
  declaring using DECLARE statement, 736
  resignaling, 936
  signaling, 992
exclusive OR
  bitwise operator, 19
EXEC statement
  Transact-SQL syntax, 796
EXECUTE
  revoking EXECUTE on a procedure using the
  REVOKE statement, 946
EXECUTE IMMEDIATE statement
  syntax, 791
EXECUTE statement
  embedded SQL syntax, 794
  Transact-SQL syntax, 796
executing
  operating system commands, 1017
  prepared statements, 794
  resuming execution of procedures, 939
  SQL statements from files, 922
  stored procedures in Transact-SQL, 796
execution plans
  example of saving a plan to a file, 253
execution time
  START LOGGING statement, 999
EXISTS search condition
  search conditions, 40
  syntax, 62
exit codes
  EXIT statement [Interactive SQL], 797
EXIT statement
  Interactive SQL syntax, 797
exiting
  Interactive SQL, 797
  procedures, 940
EXP function
  syntax, 243
EXPERIENCE_ESTIMATE function
  syntax, 243
Expires
  HTTP_RESPONSE_HEADER function, 268
EXPLAIN statement
  embedded SQL syntax, 799
EXPLANATION function
  syntax, 244
explicit selectivity estimates
  about, 64
exponential function
  EXP function, 243
exporting
  BLOBs, 1344
  unloading data using the UNLOAD statement, 1027
exporting data
  tables into files, 907
expressions
  about, 21
  arithmetic operators, 11
array operators, 18
CASE expressions, 24
column names, 23
common element in SQL syntax, 421
constants, 22
data types of, 246
IF expressions, 23
SQL operator precedence, 20
string operators, 11
subqueries, 23
syntax, 21
Transact-SQL compatibility, 39
EXPRTYPE function
  syntax, 246
extended procedures
  about, 1086
external environments
  adding comments using the COMMENT statement, 538
  starting using the START EXTERNAL ENVIRONMENT statement, 997
  stopping using the STOP EXTERNAL ENVIRONMENT statement, 1005
external functions
  interfacing to native functions using CREATE FUNCTION statement, 580
  interfacing to native procedures using CREATE PROCEDURE statement, 620
external logins
  assigning for remote servers, 578
  dropping for remote servers, 758
EXTERNAL NAME clause
  ALTER TEXT CONFIGURATION statement, 501
  CREATE FUNCTION statement [external procedures], 582
  CREATE PROCEDURE statement [external procedures], 624
external objects
  creating using the INSTALL EXTERNAL OBJECT statement, 866
  removing using the REMOVE EXTERNAL OBJECT statement, 933
  SYSEXTERNENVOBJECT system view, 1360
external procedures
  interfacing to native procedures using CREATE PROCEDURE statement, 620
external term breakers
  ALTER TEXT CONFIGURATION statement, 502
FALSE conditions
   IS FALSE search condition, 63
   three-valued logic, 63
FASTFIRSTROW table hint
   FROM clause, 818
features
   locking, 889
February 29
   about, 115
FETCH statement
   embedded SQL syntax, 800
   syntax, 800
fetching
   rows from cursors, 800
FILE clause
   DROP REMOTE MESSAGE TYPE statement, 767
   GRANT CONSOLIDATE statement [SQL Remote], 834
FILE message type
   SQL Remote ALTER REMOTE MESSAGE TYPE statement, 459
   SQL Remote CREATE REMOTE MESSAGE TYPE statement, 651
File property
   DB_EXTENDED_PROPERTY function, 212
file size
   creating events using the CREATE EVENT statement, 570
filename
   common element in SQL syntax, 421
files
   allocating space for database, 432
   creating database using the CREATE DBSPACE statement, 557
   decrypting using CREATE DECRYPTED FILE statement, 560
   decrypting with CREATE DECRYPTED DATABASE statement, 558
   encrypting using the CREATE ENCRYPTED FILE statement, 567
   exporting data from tables into, 907
   importing data into tables from, 854
   querying within files, 814
   reading on a client computer, 328
writing to a client computer, 408
xp_read_file system procedure, 1334
xp_write_file system procedure, 1344
FileSize property
   DB_EXTENDED_PROPERTY function, 212
filler() column name
   FROM clause, 814
   LOAD TABLE statement, 873
FIRST clause
   DELETE statement, 742
   FETCH statement, 801
   SELECT statement, 957
   UPDATE statement, 1039
FIRST_VALUE function
   syntax, 246
FLOAT data type
   comparing with dates and times, 134
   syntax, 101
FLOOR function
   syntax, 249
FOLLOWING clause
   WINDOW clause, 1053
FOR clause
   ALTER EVENT statement, 437
   ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 483
   CREATE EVENT statement, 573
   CREATE SUBSCRIPTION statement [SQL Remote], 683
   CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 687
   CREATE TRIGGER [Transact-SQL] statement, 720
   DECLARE CURSOR statement [T-SQL], 732
   DESCRIBE statement [ESQL], 744
   DROP SUBSCRIPTION statement [SQL Remote], 776
   DROP SYNCHRONIZATION SUBSCRIPTION statement, 778
   MESSAGE statement, 900
   SELECT statement, 960
   START SUBSCRIPTION statement [SQL Remote], 1000
   STOP SUBSCRIPTION statement [SQL Remote], 1008
   SYNCHRONIZE SUBSCRIPTION statement [SQL Remote], 1016
FOR DELETE clause
integrity constraints in CREATE TABLE
statement, 698
renaming using the ALTER INDEX statement, 441
sp_remote_exported_keys system procedure, 1300
sp_remote_imported_keys system procedure, 1302
system views, 1361
unnamed in CREATE TABLE statement, 698
foreign tables
system views, 1361
forest
in unparsed XML document, 415
FORMAT BCP clause
LOAD TABLE statement, 880
FORMAT clause
ALTER SERVICE statement [SOAP web service], 471
CREATE SERVICE statement [SOAP web service], 473, 670
INPUT statement, 856
LOAD TABLE statement, 880
OUTPUT statement, 909
UNLOAD statement, 1029
FORMAT FIXED clause
INPUT statement, 856
FORMAT SHAPEFILE clause
INPUT statement, 857
LOAD TABLE statement, 880
FORMAT TEXT clause
INPUT statement, 856
LOAD TABLE statement, 880
FORMAT XML clause
LOAD TABLE statement, 881
FORWARD TO statement
syntax, 808
fragmentation
defragmenting using REORGANIZE TABLE, 935
sa_index_density system procedure, 1160
sa_table_fragmentation, 1253
FREE PAGE ELIMINATION clause
BACKUP statement, 520
FreePages property
DB_EXTENDED_PROPERTY function, 212
frequency
sending messages, GRANT CONSOLIDATE
statement [SQL Remote], 834
sending messages, GRANT REMOTE statement
[SQL Remote], 844
FROM clause
CREATE CERTIFICATE statement, 548
CREATE DECRYPTED DATABASE statement, 559
CREATE DECRYPTED FILE statement, 560
CREATE ENCRYPTED DATABASE statement, 565
CREATE ENCRYPTED FILE statement, 567
CREATE ENCRYPTED TABLE DATABASE
statement, 565
CREATE TEXT CONFIGURATION statement, 709
DELETE statement, 742
DELETE statement (positioned) [ESQL] [SP], 739
INSTALL EXTERNAL OBJECT statement, 866
INSTALL JAVA statement, 869
LOAD TABLE statement, 875
PUT statement [ESQL], 919
RESTORE DATABASE statement, 937
REVOKE ROLE
SYS_REPLICATION_ADMIN_ROLE statement
[MobiLink] [SQL Remote], 948
REVOKE ROLE
SYS_RUN_REPLICATION_ROLE statement
[MobiLink] [SQL Remote], 949
SELECT statement, 959
selecting from DML, 815
selecting from stored procedures, 812
syntax, 810
UPDATE (positioned) statement, 1032
UPDATE statement, 1039
FROM FILE clause
INSTALL EXTERNAL OBJECT statement, 866
INSTALL JAVA statement, 868
FROM SERVER clause
ALTER MIRROR SERVER statement, 453
CREATE MIRROR SERVER statement, 617
FROM VALUE clause
INSTALL EXTERNAL OBJECT statement, 866
FTP clause
GRANT CONSOLIDATE statement [SQL Remote], 834
FTP message type
SQL Remote ALTER REMOTE MESSAGE TYPE statement, 459
SQL Remote CREATE REMOTE MESSAGE TYPE statement, 651
full text search
ALTER TEXT CONFIGURATION statement, 501
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER TEXT INDEX statement</td>
<td>504</td>
</tr>
<tr>
<td>asterisk, allowed syntax</td>
<td>59</td>
</tr>
<tr>
<td>CONTAINS clause in the FROM clause</td>
<td>810</td>
</tr>
<tr>
<td>CONTAINS search condition</td>
<td>55</td>
</tr>
<tr>
<td>CREATE TEXT CONFIGURATION statement</td>
<td>709</td>
</tr>
<tr>
<td>CREATE TEXT INDEX statement</td>
<td>711</td>
</tr>
<tr>
<td>DROP TEXT CONFIGURATION statement</td>
<td>781</td>
</tr>
<tr>
<td>DROP TEXT INDEX statement</td>
<td>782</td>
</tr>
<tr>
<td>hyphen, allowed syntax</td>
<td>60</td>
</tr>
<tr>
<td>operator precedence</td>
<td>58</td>
</tr>
<tr>
<td>REFRESH TEXT INDEX statement</td>
<td>928</td>
</tr>
<tr>
<td>sa_char_terms system procedure</td>
<td>1095</td>
</tr>
<tr>
<td>sa_nchar_terms system procedure</td>
<td>1195</td>
</tr>
<tr>
<td>sa_refresh_text_indexes system procedure</td>
<td>1211</td>
</tr>
<tr>
<td>sa_text_index_stats system procedure</td>
<td>1256</td>
</tr>
<tr>
<td>sa_text_index_vocab system procedure</td>
<td>1258</td>
</tr>
<tr>
<td>sa_text_index_vocab_nchar system procedure</td>
<td>1259</td>
</tr>
<tr>
<td>snapshot isolation compatibility</td>
<td>711</td>
</tr>
<tr>
<td>syntax for special characters</td>
<td>60</td>
</tr>
<tr>
<td>TRUNCATE TEXT INDEX statement</td>
<td>1021</td>
</tr>
<tr>
<td>warning about using non-alphanumerics in query string</td>
<td>58</td>
</tr>
<tr>
<td>functions</td>
<td></td>
</tr>
<tr>
<td>aggregate</td>
<td>143</td>
</tr>
<tr>
<td>ALTER FUNCTION statement</td>
<td>439</td>
</tr>
<tr>
<td>bit array</td>
<td>145</td>
</tr>
<tr>
<td>compared to procedures</td>
<td>149</td>
</tr>
<tr>
<td>creating SQL stored functions</td>
<td>594</td>
</tr>
<tr>
<td>creating using the CREATE FUNCTION statement [Web service]</td>
<td>586</td>
</tr>
<tr>
<td>data type conversion SQL</td>
<td>146</td>
</tr>
<tr>
<td>date and time</td>
<td>146</td>
</tr>
<tr>
<td>dropping using the DROP FUNCTION statement</td>
<td>758</td>
</tr>
<tr>
<td>exiting from user-defined</td>
<td>940</td>
</tr>
<tr>
<td>HTTP</td>
<td>151</td>
</tr>
<tr>
<td>image SQL</td>
<td>155</td>
</tr>
<tr>
<td>indexes on</td>
<td>601</td>
</tr>
<tr>
<td>interfacing to native functions using CREATE FUNCTION statement</td>
<td>580</td>
</tr>
<tr>
<td>interfacing to native procedures using CREATE PROCEDURE statement</td>
<td>620</td>
</tr>
<tr>
<td>introduction</td>
<td>143</td>
</tr>
<tr>
<td>Java</td>
<td>148</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>150</td>
</tr>
<tr>
<td>named parameters</td>
<td>87</td>
</tr>
<tr>
<td>native function calls</td>
<td>582</td>
</tr>
<tr>
<td>numeric</td>
<td>151</td>
</tr>
<tr>
<td>ranking</td>
<td>145</td>
</tr>
<tr>
<td>replacing external function interface</td>
<td>581</td>
</tr>
<tr>
<td>returning values from user-defined</td>
<td>940</td>
</tr>
<tr>
<td>SOAP</td>
<td>151</td>
</tr>
<tr>
<td>string</td>
<td>152</td>
</tr>
<tr>
<td>system</td>
<td>154</td>
</tr>
<tr>
<td>temporary</td>
<td>595</td>
</tr>
<tr>
<td>text SQL</td>
<td>155</td>
</tr>
<tr>
<td>types of functions</td>
<td>143</td>
</tr>
<tr>
<td>user defined</td>
<td>148</td>
</tr>
<tr>
<td>functions, aggregate</td>
<td></td>
</tr>
<tr>
<td>about</td>
<td>143</td>
</tr>
<tr>
<td>AVG</td>
<td>164</td>
</tr>
<tr>
<td>BIT_AND</td>
<td>167</td>
</tr>
<tr>
<td>BIT_OR</td>
<td>169</td>
</tr>
<tr>
<td>BIT_XOR</td>
<td>171</td>
</tr>
<tr>
<td>COUNT</td>
<td>192</td>
</tr>
<tr>
<td>COUNT_BIG</td>
<td>193</td>
</tr>
<tr>
<td>FIRST_VALUE</td>
<td>246</td>
</tr>
<tr>
<td>GROUPING</td>
<td>255</td>
</tr>
<tr>
<td>LAST_VALUE</td>
<td>279</td>
</tr>
<tr>
<td>LIST</td>
<td>285</td>
</tr>
<tr>
<td>MAX</td>
<td>292</td>
</tr>
<tr>
<td>MEDIAN</td>
<td>293</td>
</tr>
<tr>
<td>MIN</td>
<td>295</td>
</tr>
<tr>
<td>SET_BITS</td>
<td>360</td>
</tr>
<tr>
<td>STDDEV</td>
<td>372</td>
</tr>
<tr>
<td>STDDEV_POP</td>
<td>372</td>
</tr>
<tr>
<td>STDDEV_SAMP</td>
<td>373</td>
</tr>
<tr>
<td>SUM</td>
<td>381</td>
</tr>
<tr>
<td>VAR_POP</td>
<td>402</td>
</tr>
<tr>
<td>VAR_SAMP</td>
<td>404</td>
</tr>
<tr>
<td>VARIANCE</td>
<td>406</td>
</tr>
<tr>
<td>functions, bit array</td>
<td></td>
</tr>
<tr>
<td>about</td>
<td>145</td>
</tr>
<tr>
<td>alphabetical list</td>
<td>145</td>
</tr>
<tr>
<td>BIT_LENGTH</td>
<td>168</td>
</tr>
<tr>
<td>BIT_SUBSTR</td>
<td>169</td>
</tr>
<tr>
<td>COUNT_SET_BITS</td>
<td>195</td>
</tr>
<tr>
<td>SET_BIT</td>
<td>359</td>
</tr>
<tr>
<td>functions, data type conversion</td>
<td></td>
</tr>
<tr>
<td>about</td>
<td>146</td>
</tr>
<tr>
<td>BINTOHEX</td>
<td>166</td>
</tr>
<tr>
<td>CAST</td>
<td>174</td>
</tr>
</tbody>
</table>
functions, date and time
about, 146
DATE, 203
DATEADD, 204
DATEDIFF, 205
DATEFORMAT, 206
DATENAME, 207
DATEPART, 208
DATETIME, 209
DAY, 210
DAYNAME, 210
DAYS, 211
DOW, 225
GETDATE, 251
HOUR, 259
HOURS, 259
MINUTE, 296
MINUTES, 297
MONTH, 299
MONTHNAME, 300
MONTHS, 301
NOW, 311
QUARTER, 324
SECOND, 356
SECONDS, 357
SWITCHOFFSET, 383
SYSDATETIMEOFFSET, 384
TODATETIMEOFFSET, 388
TODAY, 389
WEEKS, 407
YEAR, 417
YEARS, 417
YMD, 419
functions, HTTP
HTTP_HEADER, 266
HTTP_RESPONSE_HEADER, 268
HTTP_VARIABLE, 270
NEXT_HTTP_HEADER, 306
NEXT_HTTP_RESPONSE_HEADER, 308
NEXT_HTTP_VARIABLE, 309
functions, Java and SQL user defined
about, 148
functions, miscellaneous
about, 150
ARGN, 157
COALESCE, 179
CONFLICT, 182
ERRORMSG, 235
ESTIMATE, 236
ESTIMATE_SOURCE, 237
EXPERIENCE_ESTIMATE, 243
EXPLANATION, 244
GET_IDENTITY, 250
GRAPHICAL_PLAN, 252
GREATER, 254
IDENTITY, 271
IDENTITY, 271
INDEX_ESTIMATE, 273
ISNUMERIC, 278
LESSER, 284
NEWID, 303
NULLIF, 312
NUMBER, 313
PLAN, 318
REWRITE, 348
SQLDIALECT, 368
TRACEBACK, 389
TRACED_PLAN, 390
TRANSACTSQL, 391
VAREXISTS, 406
WATCOMSQL, 406
functions, numeric
about, 151
ABS, 155
ACOS, 156
ASIN, 161
ATAN, 162
ATAN2, 163
ATN2, 163
CEILING, 175
COT, 191
COS, 191
DEGREES, 222
EXP, 243
FLOOR, 249
LOG, 289
LOG10, 289
MOD, 299
PI, 318
POWER, 319
fuzzy searches
  CONTAINS search condition, 55

G
g-geometries
  troubleshooting using st_geometry_dump, 1322
GET DATA statement
  embedded SQL syntax, 822
GET_DESCRIPTOR statement
  embedded SQL syntax, 824
GET OPTION statement
  embedded SQL syntax, 825
GET_BIT function
  syntax, 249
GET.IDENTITY function
  syntax, 250
GETDATE function
  query the current system date, 252
  syntax, 251
getting
  binary data from columns, 822
  information from descriptor areas, 824
  option values, 825
GLOBAL AUTOINCREMENT
  creating events using the CREATE EVENT statement, 570
GLOBAL AUTOINCREMENT clause
  CREATE TABLE statement, 690
GLOBAL AUTOINCREMENT default
  CREATE TABLE statement, 490, 696
global checksums
  altering database settings, 428
  creating databases with, 551
  validating, 1045
global temporary tables
  CREATE TABLE statement, 690
global variables
  @@identity, 85
  alphabetical list, 82
  definition, 79
  triggers and @@identity, 85
global_database_id option
  CREATE TABLE statement, 490, 696
globally unique identifiers
  SQL syntax for NEWID function, 303
goodness of fit
  regression lines, 337
GOTO clause
  WHENEVER statement [ESQL], 1049
GOTO statement
  Transact-SQL syntax, 827
GRANT CONNECT statement
  character set conversion on passwords, 833
  password maximum length, 833
  passwords, character set conversion on, 833
  syntax, 833
GRANT CONSOLIDATE statement
  SQL Remote syntax, 834
GRANT CREATE statement
  syntax, 836
GRANT EXECUTE statement
  syntax, 836
GRANT INTEGRATED LOGIN statement
  syntax, 837
GRANT KERBEROS LOGIN statement
  syntax, 838
GRANT PUBLISH statement
  SQL Remote syntax, 839
GRANT REMOTE statement
  SQL Remote syntax, 844
GRANT ROLE statement
  syntax, 827
GRANT ROLE
  SYS_REPLICATION_ADMIN_ROLE statement
    MobiLink syntax, 840
    SQL Remote syntax, 840
GRANT ROLE SYS_RUN_REPLICATION_ROLE statement
  MobiLink syntax, 842
  SQL Remote syntax, 842
GRANT SET USER statement
  syntax, 827
GRANT statement
  reviewing privileges, 1351
  syntax, 827
GRANT USAGE ON SEQUENCE statement
  syntax, 846
granting
  CONSOLIDATE privilege, 834
  impersonation (SET USER system privilege), 827
  object-level privileges, 827
  privileges, 827
  PUBLISH privilege, 839
  REMOTE privilege, 844
GRAPHICAL_PLAN function
Index

syntax, 252
GREATER function
  syntax, 254
greater than
  comparison operator, 9
greater than or equal to
  comparison operator, 9
grid size
  ALTER SPATIAL REFERENCE SYSTEM
    statement, 479
  CREATE SPATIAL REFERENCE SYSTEM
    statement, 678
GROUP BY clause
  CUBE operation, 848
  GROUPING SETS operation, 847
  ROLLUP operation, 847
  SELECT statement, 959
  syntax, 847
GROUP clause
  ALTER SERVICE statement [SOAP web service],
    471
  CREATE SERVICE statement [SOAP web service], 473, 669
grouping
  GROUP BY clause, 847
  regular expressions, 27
  statements in a BEGIN statement, 523
  statements in a BEGIN statement [TSQL], 526
  statements in a TRY statement, 1022
GROUPING function
  syntax, 255
GROUPING SETS operation
  GROUP BY clause, 847
GUIDs
  SQL syntax for NEWID function, 303
  SQL syntax for STRTOUID function, 377
  SQL syntax for UUIDTOSTR function, 401
  UNIQUEIDENTIFIER data type, 126
gzip utility
  COMPRESS function, 181
  DECOMPRESS function, 219

H
HANDLER clause
  CREATE EVENT statement, 574
handling
  errors in embedded SQL, 1049
  RAISERROR statement, 920
  HASH function
    syntax, 255
  hashing
    supported algorithms, 255
  HAVING clause
    search conditions, 40
    SELECT statement, 960
  HEADER clause
    CREATE FUNCTION statement [web clients],
      588
    CREATE PROCEDURE statement [web clients],
      631
  HELP statement
    Interactive SQL syntax, 850
  hexadecimal
    converting to and from hexadecimal values, 6
    converting using CAST, CONVERT, HEXTOINT, and INTTOHEX functions, 6
  HEXADECIMAL clause
    LOAD TABLE statement, 882
    OUTPUT statement, 909
    UNLOAD statement, 1029
  hexadecimal constants
    converting to and from hexadecimal values, 6
    treated as binary, 6
  hexadecimal escape sequences
    in SQL strings, 8
  hexadecimal strings
    about, 166, 257, 258
  HEXTOBIN function
    syntax, 257
  HEXTOINT function
    syntax, 258
  high availability
    ALTER MIRROR SERVER statement, 452
    CREATE MIRROR SERVER statement, 615
  hinting
    index hinting using the FROM clause, 818
  histograms
    creating using CREATE STATISTICS, 682
    only partially updated by LOAD TABLE, 886
    retrieving, 1145
    selectivity estimates, 64
    SYSCOLSTAT system view, 1351
    updating using CREATE STATISTICS, 682
  HISTORY clause
    ATTACH TRACING statement, 514
BACKUP statement, 518
RESTORE DATABASE statement, 938
HOLDLOCK clause
  READTEXT statement [T-SQL], 924
HOLDLOCK table hint
  FROM clause, 816
Host
  HTTP_HEADER function, 266
host variables
  common element in SQL syntax, 421
declaring in embedded SQL, 728
hostvar
  common element in SQL syntax, 421
HOUR function
  syntax, 259
HOURS function
  syntax, 259
HTML
  CREATE SERVICE statement [HTTP web service], 466, 663
HTML_DECODE function
  syntax, 261
HTML_ENCODE function
  syntax, 262
HTTP
  setting headers, 1239
  setting options, 1240
  setting options for SOAP headers, 1244
HTTP functions
  alphabetical list, 151
HTTP headers
  HTTP_HEADER function, 266
  HTTP_RESPONSE_HEADER function, 268
HTTP message type
  SQL Remote CREATE REMOTE MESSAGE TYPE statement, 651
HTTP request headers
  @HttpMethod, 267
  @HttpQueryString, 267
  @HttpURI, 267
  @HttpVersion, 267
  Accept-Encoding, 266
  Cookie, 266
  Host, 266
  Referer, 266
  returning names and values, 1154
  User-Agent, 266
HTTP response headers
  @HttpStatus, 269
  Connection, 268
  Content-Length, 268
  Content-Type, 268
  Date, 268
  Expires, 268
  Location, 268
  sa_set_http_header_info system procedure, 1239
  Server, 268
  Transfer-Encoding, 268
  User-Agent, 268
  WWW-Authenticate, 268
HTTP services
  creating using CREATE SERVICE statement [HTTP web service], 662
HTTP system procedures
  alphabetical list, 1085
HTTP_BODY function
  syntax, 263
HTTP_DECODE function
  syntax, 264
HTTP_ENCODE function
  syntax, 264
HTTP_HEADER function
  syntax, 266
HTTP_RESPONSE_HEADER function
  syntax, 268
HTTP_VARIABLE function
  syntax, 270
HttpMethod
  HTTP_HEADER function, 267
HttpQuerystring
  HTTP_HEADER function, 267
HttpStatus header
  HTTP_RESPONSE_HEADER function, 269
  sa_set_http_header_info system procedure, 1239
HttpURI
  HTTP_HEADER function, 267
HttpVersion
  HTTP_HEADER function, 267
hyphens
  allowed syntax in a CONTAINS clause, 60
  allowed syntax in a full text query string, 60

I/O
  recalibrating the I/O cost model, 429
<table>
<thead>
<tr>
<th>Phrase</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENTIFIED BY clause</td>
<td></td>
</tr>
<tr>
<td>ALTER LDAP SERVER statement</td>
<td>443</td>
</tr>
<tr>
<td>ALTER Spatial Reference System statement</td>
<td>478</td>
</tr>
<tr>
<td>CREATE EXTERNLOGIN statement</td>
<td>579</td>
</tr>
<tr>
<td>CREATE LDAP SERVER statement</td>
<td>604</td>
</tr>
<tr>
<td>CREATE Spatial Reference System statement</td>
<td>674</td>
</tr>
<tr>
<td>CREATE USER statement</td>
<td>721</td>
</tr>
<tr>
<td>IDENTIFIED BY ENCRYPTED clause</td>
<td></td>
</tr>
<tr>
<td>ALTER LDAP SERVER statement</td>
<td>443</td>
</tr>
<tr>
<td>CREATE LDAP SERVER statement</td>
<td>604</td>
</tr>
<tr>
<td>IDENTIFIED BY password clause</td>
<td></td>
</tr>
<tr>
<td>ALTER USER statement</td>
<td>509</td>
</tr>
<tr>
<td>CONNECT statement</td>
<td>543</td>
</tr>
<tr>
<td>identifiers</td>
<td></td>
</tr>
<tr>
<td>about</td>
<td>4</td>
</tr>
<tr>
<td>maximum length in SQL Anywhere</td>
<td>4</td>
</tr>
<tr>
<td>syntax</td>
<td>4</td>
</tr>
<tr>
<td>using in domains</td>
<td>564</td>
</tr>
<tr>
<td>IDENTIFY clause</td>
<td></td>
</tr>
<tr>
<td>ALTER TABLE statement</td>
<td>493</td>
</tr>
<tr>
<td>CREATE TABLE statement</td>
<td>697</td>
</tr>
<tr>
<td>IDENTIFY column</td>
<td></td>
</tr>
<tr>
<td>@identity</td>
<td>85</td>
</tr>
<tr>
<td>IDENTIFY function</td>
<td></td>
</tr>
<tr>
<td>syntax</td>
<td>271</td>
</tr>
<tr>
<td>idle server</td>
<td></td>
</tr>
<tr>
<td>creating events using the CREATE EVENT statement</td>
<td>570</td>
</tr>
<tr>
<td>IdleTime event condition</td>
<td></td>
</tr>
<tr>
<td>about</td>
<td>238</td>
</tr>
<tr>
<td>IdleTimeout property</td>
<td></td>
</tr>
<tr>
<td>setting with sa_server_option</td>
<td>1222</td>
</tr>
<tr>
<td>IF EXISTS clause</td>
<td></td>
</tr>
<tr>
<td>DROP EVENT statement</td>
<td>757</td>
</tr>
<tr>
<td>DROP EXTERNLOGIN statement</td>
<td>758</td>
</tr>
<tr>
<td>DROP INDEX statement</td>
<td>759</td>
</tr>
<tr>
<td>DROP MATERIALIZED VIEW statement</td>
<td>762</td>
</tr>
<tr>
<td>DROP PROCEDURE statement</td>
<td>765</td>
</tr>
<tr>
<td>DROP PUBLICATION statement [MobiLink] [SQL Remote]</td>
<td>766</td>
</tr>
<tr>
<td>DROP Spatial Reference System statement</td>
<td>773</td>
</tr>
<tr>
<td>DROP Spatial Unit Of Measure statement</td>
<td>774</td>
</tr>
<tr>
<td>DROP SYNCHRONIZATION PROFILE statement</td>
<td>778</td>
</tr>
<tr>
<td>DROP TABLE statement</td>
<td>780</td>
</tr>
<tr>
<td>DROP TRIGGER statement</td>
<td>785</td>
</tr>
<tr>
<td>DROP VARIABLE statement</td>
<td>787</td>
</tr>
<tr>
<td>DROP VIEW statement</td>
<td>788</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phrase</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF expressions</td>
<td></td>
</tr>
<tr>
<td>search conditions</td>
<td>40</td>
</tr>
<tr>
<td>syntax</td>
<td>23</td>
</tr>
<tr>
<td>IF NOT EXISTS clause</td>
<td></td>
</tr>
<tr>
<td>CREATE INDEX statement</td>
<td>600</td>
</tr>
<tr>
<td>CREATE LOCAL TEMPORARY TABLE statement</td>
<td>606</td>
</tr>
<tr>
<td>CREATE PUBLICATION statement [MobiLink] [SQL Remote]</td>
<td>648</td>
</tr>
<tr>
<td>CREATE TABLE statement</td>
<td>693</td>
</tr>
<tr>
<td>CREATE TEXT INDEX statement</td>
<td>711</td>
</tr>
<tr>
<td>IF statement</td>
<td></td>
</tr>
<tr>
<td>syntax</td>
<td>850</td>
</tr>
<tr>
<td>Transact-SQL syntax</td>
<td>852</td>
</tr>
<tr>
<td>IF UPDATE clause</td>
<td></td>
</tr>
<tr>
<td>CREATE TRIGGER [Transact-SQL] statement</td>
<td>720</td>
</tr>
<tr>
<td>in triggers</td>
<td>713</td>
</tr>
<tr>
<td>in triggers in Transact-SQL</td>
<td>720</td>
</tr>
<tr>
<td>IFNULL function</td>
<td></td>
</tr>
<tr>
<td>syntax</td>
<td>272</td>
</tr>
<tr>
<td>image backups</td>
<td></td>
</tr>
<tr>
<td>creating using the BACKUP statement</td>
<td>516</td>
</tr>
<tr>
<td>IMAGE data type</td>
<td></td>
</tr>
<tr>
<td>syntax</td>
<td>125</td>
</tr>
<tr>
<td>image SQL functions</td>
<td></td>
</tr>
<tr>
<td>about</td>
<td>155</td>
</tr>
<tr>
<td>images</td>
<td></td>
</tr>
<tr>
<td>reading from the database</td>
<td>924</td>
</tr>
<tr>
<td>IMMEDIATE REFRESH clause</td>
<td></td>
</tr>
<tr>
<td>ALTER MATERIALIZED VIEW statement</td>
<td>451</td>
</tr>
<tr>
<td>CREATE MATERIALIZED VIEW statement</td>
<td>612</td>
</tr>
<tr>
<td>CREATE TEXT INDEX statement</td>
<td>711</td>
</tr>
<tr>
<td>immediate views</td>
<td></td>
</tr>
<tr>
<td>ALTER MATERIALIZED VIEW statement</td>
<td>450</td>
</tr>
<tr>
<td>privileges not required to update underlying tables</td>
<td>451</td>
</tr>
<tr>
<td>impersonation</td>
<td></td>
</tr>
<tr>
<td>SETUSER statement</td>
<td>990</td>
</tr>
<tr>
<td>importing data</td>
<td></td>
</tr>
<tr>
<td>binary files</td>
<td>1334</td>
</tr>
<tr>
<td>images</td>
<td>1334</td>
</tr>
<tr>
<td>into tables from files</td>
<td>854</td>
</tr>
<tr>
<td>IN clause</td>
<td></td>
</tr>
<tr>
<td>SQL Statement</td>
<td>Page(s)</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>CREATE FUNCTION statement</td>
<td>594</td>
</tr>
<tr>
<td>CREATE FUNCTION statement [Web service]</td>
<td>586</td>
</tr>
<tr>
<td>CREATE MATERIALIZED VIEW statement</td>
<td>612</td>
</tr>
<tr>
<td>CREATE PROCEDURE statement</td>
<td>640</td>
</tr>
<tr>
<td>CREATE TABLE statement</td>
<td>692</td>
</tr>
<tr>
<td>CREATE TEXT INDEX statement</td>
<td>711</td>
</tr>
<tr>
<td>LOCK TABLE statement</td>
<td>890</td>
</tr>
<tr>
<td>IN EXCLUSIVE MODE clause LOCK TABLE statement</td>
<td>890</td>
</tr>
<tr>
<td>IN search condition syntax</td>
<td>54</td>
</tr>
<tr>
<td>IN SHARE MODE clause LOCK TABLE statement</td>
<td>890</td>
</tr>
<tr>
<td>IN</td>
<td>ON clause CREATE INDEX statement</td>
</tr>
<tr>
<td>INCLUDE statement embedded SQL syntax</td>
<td>853</td>
</tr>
<tr>
<td>INCREMENT BY clause ALTER SEQUENCE statement</td>
<td>462</td>
</tr>
<tr>
<td>CREATE SEQUENCE statement</td>
<td>656</td>
</tr>
<tr>
<td>independent variables regression line</td>
<td>331</td>
</tr>
<tr>
<td>INDEX clause ALTER INDEX statement</td>
<td>441</td>
</tr>
<tr>
<td>ALTER TABLE statement</td>
<td>492</td>
</tr>
<tr>
<td>CREATE TABLE statement</td>
<td>694</td>
</tr>
<tr>
<td>FROM clause</td>
<td>818</td>
</tr>
<tr>
<td>REORGANIZE TABLE statement</td>
<td>935</td>
</tr>
<tr>
<td>VALIDATE statement</td>
<td>1045</td>
</tr>
<tr>
<td>INDEX FOR clause DESCRIBE statement</td>
<td>748</td>
</tr>
<tr>
<td>index hints FROM clause</td>
<td>818</td>
</tr>
<tr>
<td>INDEX ONLY clause FROM clause</td>
<td>818</td>
</tr>
<tr>
<td>INDEX_COL function syntax</td>
<td>154</td>
</tr>
<tr>
<td>INDEX_ESTIMATE function syntax</td>
<td>273</td>
</tr>
<tr>
<td>indexes adding comments using the COMMENT statement</td>
<td>538</td>
</tr>
<tr>
<td>ALTER INDEX statement</td>
<td>441</td>
</tr>
<tr>
<td>automatically created</td>
<td>602</td>
</tr>
<tr>
<td>built-in functions</td>
<td>599</td>
</tr>
<tr>
<td>clustering using the ALTER INDEX statement</td>
<td>441</td>
</tr>
<tr>
<td>compressing</td>
<td>935</td>
</tr>
<tr>
<td>CREATE INDEX statement</td>
<td>599</td>
</tr>
<tr>
<td>detecting index fragmentation using sa_index_density</td>
<td>1160</td>
</tr>
<tr>
<td>detecting skewed indexes using sa_index_density</td>
<td>1160</td>
</tr>
<tr>
<td>DROP INDEX statement</td>
<td>759</td>
</tr>
<tr>
<td>foreign keys</td>
<td>602</td>
</tr>
<tr>
<td>functions</td>
<td>602</td>
</tr>
<tr>
<td>levels</td>
<td>1162</td>
</tr>
<tr>
<td>naming</td>
<td>602</td>
</tr>
<tr>
<td>optimizing for OLAP workloads</td>
<td>601</td>
</tr>
<tr>
<td>owner</td>
<td>602</td>
</tr>
<tr>
<td>physical indexes recorded in SYSPHYSIDX system view</td>
<td>1376</td>
</tr>
<tr>
<td>primary keys</td>
<td>602</td>
</tr>
<tr>
<td>renaming using the ALTER INDEX statement</td>
<td>441</td>
</tr>
<tr>
<td>system views</td>
<td>1364</td>
</tr>
<tr>
<td>table use</td>
<td>602</td>
</tr>
<tr>
<td>unique</td>
<td>600</td>
</tr>
<tr>
<td>VALIDATE statement</td>
<td>1045</td>
</tr>
<tr>
<td>views</td>
<td>602</td>
</tr>
<tr>
<td>virtual</td>
<td>600</td>
</tr>
<tr>
<td>INDICATOR clause GET DESCRIPTOR statement [ESQL]</td>
<td>824</td>
</tr>
<tr>
<td>SET DESCRIPTOR statement [ESQL]</td>
<td>967</td>
</tr>
<tr>
<td>indicators comments</td>
<td>85</td>
</tr>
<tr>
<td>initializing databases using the CREATE DATABASE statement</td>
<td>549</td>
</tr>
<tr>
<td>INLINE clause ALTER SEQUENCE statement</td>
<td>492</td>
</tr>
<tr>
<td>CREATE TABLE statement</td>
<td>693</td>
</tr>
<tr>
<td>INNER JOIN clause FROM clause SQL syntax</td>
<td>810</td>
</tr>
<tr>
<td>INOUT clause CREATE PROCEDURE statement</td>
<td>640</td>
</tr>
<tr>
<td>INPUT clause DESCRIBE statement</td>
<td>745</td>
</tr>
<tr>
<td>INPUT INTO statement Interactive SQL syntax</td>
<td>854</td>
</tr>
<tr>
<td>INPUT statement cannot be used in stored procedures</td>
<td>859</td>
</tr>
<tr>
<td>Interactive SQL syntax</td>
<td>854</td>
</tr>
<tr>
<td>syntax</td>
<td>854</td>
</tr>
<tr>
<td>INSENSITIVE clause DECLARE CURSOR statement</td>
<td>730</td>
</tr>
</tbody>
</table>
Index

FOR statement, 805

INSERT clause
CREATE PUBLICATION statement [MobiLink]
[SQL Remote], 647
CREATE TRIGGER statement, 714

INSERT privilege
GRANT statement, 828
REVOKE statement, 946

INSERT privilege clause
GRANT statement, 830

INSERT statement
multi-row inserts, 861
setting database options, 863
syntax, 860
 updating views, 864

inserting
BLOBs using the SET statement, 985
data using LOAD TABLE statement, 873
multi-row, 794
rows in bulk, 873
rows into tables, 860
rows using cursors, 919
wide inserts, 794

inserting BLOBs
using xp_read_file system procedure, 1334

INSERTSTR function
syntax, 273
INSTALL EXTERNAL OBJECT statement
syntax, 866
INSTALL JAVA statement
installing Java classes, 868
syntax, 868

installing
Java classes, 868

INSTEAD OF clause
CREATE TRIGGER statement, 713

INSTEAD OF triggers
altering views, 512
CREATE TRIGGER statement, 713
creating views, 724
dropped by CREATE OR REPLACE VIEW, 724
replacing views, 724

INTEGER data type
comparing with dates and times, 134
syntax, 102

INTEGER UNSIGNED data type
syntax, 102
integers

generating table of, 1217

Integrated logins
adding comments using the COMMENT statement, 538
GRANT statement, 837
REVOKE statement, 944

Integrity
constraints in CREATE TABLE statement, 697

Interactive SQL
BYE statement syntax, 797
CLEAR statement syntax, 537
CONFIGURE statement syntax, 543
CONNECT statement syntax, 543
DESCRIBE CONNECTION statement syntax, 747
DESCRIBE statement syntax, 747
DISCONNECT statement syntax, 751
EXIT statement syntax, 797
HELP statement syntax, 850
INPUT statement syntax, 854
OUTPUT statement, 907
PARAMETERS statement syntax, 912
procedure profiling, 1222
QUIT statement syntax, 797
READ statement syntax, 922
RESUME statement unsupported, 940
return codes, 797
SET CONNECTION statement syntax, 966
SET OPTION statement syntax, 975
specifying encoding for INPUT statement, 855
specifying encoding for OUTPUT statement, 908
specifying encoding for READ statement, 922
START DATABASE statement, 993
START LOGGING statement syntax, 999
START SERVER statement syntax, 996
STOP LOGGING statement syntax, 1007
SYSTEM statement syntax, 1017

Internal storage format
ALTER SPATIAL REFERENCE SYSTEM
statement, 479
CREATE SPATIAL REFERENCE SYSTEM
statement, 679

International languages and character sets
substitution characters, 132

INTERSECT ALL clause
INTERSECT statement, 870

INTERSECT clause
INTERSECT statement, 870

setting database options, 870
INTERSECT DISTINCT clause
INTERSECT statement, 870
INTERSECT statement
syntax, 870
intersecting
result of multiple SELECT statements, 870
Interval event condition
about, 238
INTO clause
EXECUTE statement, 794
EXPLAIN statement [ESQL], 799
FETCH statement, 801
GET DATA statement, 822
GET OPTION statement [ESQL], 825
INPUT statement, 858
MERGE statement, 894
OUTPUT statement, 907
PUT statement [ESQL], 919
SELECT statement, 958
INTO CLIENT FILE clause
UNLOAD statement, 1028
INTO DESCRIPTOR clause
PUT statement [ESQL], 919
INTO FILE clause
UNLOAD statement, 1028
INTO LOCAL TEMPORARY TABLE clause
SELECT statement, 959
INTO VARIABLE clause
UNLOAD statement, 1028
INTTOHEX function
syntax, 274
inverse flattening
defined, 1392
invoker
ALTER DATABASE statement, SYSTEM
PROCEDURE AS clause, 426
ALTER PROCEDURE statement, SQL
SECURITY clause, 639
CREATE DATABASE statement, 639
invoking
procedures using the CALL statement, 530
IOParallelism property
DB_EXTENDED_PROPERTY function, 212
IPAddressMonitorPeriod property
setting with sa_server_option, 1222
IS
logical operators description, 10
three-valued logic, 63
IS DISTINCT FROM search condition
syntax, 44
IS FALSE search condition
syntax, 63
IS NOT DISTINCT FROM search condition
syntax, 44
IS NOT NULL search condition
syntax, 62
IS NOT OF expressions
search conditions, 40
IS NULL search condition
syntax, 62
IS OF expressions
search conditions, 40
IS OF type-expression
search conditions, 41
IS TRUE search condition
syntax, 63
IS UNKNOWN search condition
syntax, 63
ISDATE function
syntax, 275
ISENCRIPTED function
syntax, 276
ISNULL function
syntax, 277
ISNUMERIC function
syntax, 278
ISO 8601
support, 109
isolation level
clause, 904
ISOLATION LEVEL clause
OPEN statement, 904
isolation levels
ISOLATION LEVEL clause, 904
table hints, 816
isolation_level option
overriding in a MERGE statement, 897
setting for DELETE statements, 743
setting for EXCEPT statement, 790
setting for INSERT statements, 863
setting for INTERSECT statement, 870
setting for UNION statement, 1026
setting for UPDATE statements, 1040
ISYSARTICLE
system table, 1055
ISYSARTICLECOL
system table, 1056
ISYSATTRIBUTE
system table, 1056
ISYSATTRIBUTENAME
system table, 1056
ISYSCAPABILITY
system table, 1056
ISYSCERTIFICATE
system table, 1056
ISYSCHECK
system table, 1056
ISYSCOLPERM
system table, 1057
ISYSCOLSTAT
encrypted if database is encrypted, 1057
encrypted if table encryption enabled, 1057
loading statistics, 873
system table, 1057
ISYSCONSTRAINT
system table, 1057
ISYSDBFILE
system table, 1057
ISYSDBSPACE
system table, 1058
ISYSDBSPACEPERM
system table, 1058
ISYSDEPENDENCY
system table, 1057
ISYSDOMAIN
system table, 1058
ISYSEVENT
system table, 1058
ISYSEXTERNENV
system table, 1058
ISYSEXTERNENVOBJECT
system table, 1058
ISYSEXTERNLOGIN
encrypted if database is encrypted, 1059
encrypted if table encryption enabled, 1059
system table, 1059
ISYSPFKEY
system table, 1059
ISYSHISTORY
system table, 1059
ISYSIDX
system table, 1059
ISYSEXIDCOL
system table, 1059
ISYSLINK
system table, 1059
ISYSLINKOBJECT
system table, 1059
ISYSLINKOBJECTPERM
system table, 1059
ISYSLINKOBJECTSID
system table, 1059
ISYSLINKSEGMENT
system table, 1059
ISYSLINKSEGMENTPERM
system table, 1059
ISYSLINKSEGMENTSID
system table, 1059
ISYSLINKSEGMENTSTAT
system table, 1059
ISYSLINKSTAT
system table, 1059
ISYSLINKSTATOBJ
system table, 1059
ISYSLINKSTATOBJPERM
system table, 1059
ISYSLINKSTATOBJSID
system table, 1059
ISYSLINKSTATSEGMENT
system table, 1059
ISYSLINKSTATSEGMENTPERM
system table, 1059
ISYSLINKSTATSEGMENTSID
system table, 1059
ISYSLINKSTATSEGMENTSTAT
system table, 1059
ISYSLINKSTATSEGMENTVALUE
system table, 1059
ISYSLINKSTATVALUE
system table, 1059
ISYSLINKSTATVALUEOBJ
system table, 1059
ISYSLINKSTATVALUEOBJPERM
system table, 1059
ISYSLINKSTATVALUEOBJSID
system table, 1059
ISYSLINKSTATVALUESEGMENT
system table, 1059
ISYSLINKSTATVALUESEGMENTPERM
system table, 1059
ISYSLINKSTATVALUESEGMENTSID
system table, 0159
ISYSLINKSTATVALUESEGMENTSTAT
system table, 1059
ISYSLINKSTATVALUESEGMENTVALUE
system table, 1059
ISYSLINKSTATVALUEVALUE
system table, 1059
ISYSLINKSTATVALUEVALUEOBJ
system table, 1059
ISYSLINKSTATVALUEVALUEOBJPERM
system table, 1059
ISYSLINKSTATVALUEVALUEOBJSID
system table, 1059
ISYSLINKSTATVALUEVALUESEGMENT
system table, 1059
ISYSLINKSTATVALUEVALUESEGMENTPERM
system table, 1059
ISYSLINKSTATVALUEVALUESEGMENTSID
system table, 1059
ISYSLINKSTATVALUEVALUESEGMENTSTAT
system table, 1059
ISYSLINKSTATVALUEVALUEVALUE
system table, 1059
ISYSLINKSTATVALUEVALUEVALUEOBJ
system table, 1059
ISYSLINKSTATVALUEVALUEVALUEOBJPERM
system table, 1059
ISYSLINKSTATVALUEVALUEVALUEOBJSID
system table, 1059
ISYSLINKSTATVALUEVALUEVALUESEGMENT
system table, 1059
ISYSLINKSTATVALUEVALUEVALUESEGMENTPERM
system table, 1059
ISYSLINKSTATVALUEVALUEVALUESEGMENTSID
system table, 1059
ISYSJAR
system table, 1060
ISYSJARCOMPONENT
system table, 1060
ISYSJAVACLASS
system table, 1060
ISYSLDAPSERVER
system table, 1060
ISYSLOGINMAP
system table, 1060
ISYSLOGINPOLICY
system table, 1061
ISYSLOGINPOLICYOPTION
system table, 1061
ISYSMIRROROROPTION
system table, 1061
ISYSMIRRORORSERVER
system table, 1061
ISYSMIRRORORSERVEROPTION
system table, 1061
ISYSMVOPTION
system table, 1061
ISYSMVOPTIONNAME
system table, 1062
ISYSOBJECT
system table, 1062
ISYSOPTION
system table, 1062
ISYSOPTSTAT
system table, 1062
ISYSPHYSIDX
system table, 1062
ISYSPROCPARM
system table, 1063
ISYSPROCPERM
system table, 1063
ISYSPROXYTAB
system table, 1063
ISYSPUBLICATION
system table, 1063
ISYSREMARK
system table, 1063
ISYSSMOTEOPTION
system table, 1064
ISYSSMOTEOPTIONTYPE
system table, 1064
ISYSREMTYPETYPE
  system table, 1064
ISYSREMOTETYPE
  system table, 1064
ISYSROLEGRANT
  system table, 1064
ISYSROLEGRANTEXT
  system table, 1065
ISYSSCHEDULE
  system table, 1065
ISYSSERVICE
  adding servers, 657
  remote servers for Component Integration Services, 657
  system table, 1065
ISYSSOURCE
  system table, 1065
ISYSSPATIALREFERENCESYSTEM
  system table, 1066
ISYSSQLEVENT
  system table, 1066
ISYSSUBSCRIPTION
  system table, 1066
ISYSSYNC
  system table, 1066
ISYSSYNCPROFILE
  system table, 1066
ISYSSYSCONFIG
  system table, 1067
ISYSTABLE
  system table, 1067
ISYSTABLECOL
  system table, 1067
ISYSTABPERMPERM
  system table, 1068
ISYSTEVENTCONF
  system table, 1067
ISYSTEVENTIDX
  system table, 1067
ISYSTEVTIDX
  system table, 1067
ISYSTABPERMPERM
  system table, 1068
ISYTTEXTCONFIG
  system table, 1067
ISYTTEXTIDX
  system table, 1067
ISYTTEXTIDXTAB
  system table, 1067
ISYSTRIGGER
  system table, 1068
ISYSTYPEMAP
  system table, 1068
ISYSUNITOFMEASURE
  system table, 1068
ISYSUSER
  encrypted if database is encrypted, 1068
  encrypted if table encryption enabled, 1068
  system table, 1068
ISYSUSERSMESSAGE
  system table, 1069
ISYSUSERTYPE
  system table, 1069
ISYSSCRIPT
  system table, 1069
ISYSSCHEDULE
  system table, 1069
ISYSSERVICES
  adding HTTP services using CREATE SERVICE statement [HTTP web service], 662
  adding SOAP over HTTP services, 668
  altering HTTP web services, 466
  altering SOAP over HTTP services, 471
  system table, 1069
iterating
  over cursors, 805

J

JAR clause
  INSTALL JAVA statement, 868
  REMOVE JAVA statement, 934
JAR files
  adding comments using the COMMENT statement, 538
  INSTALL JAVA statement, 868
  removing, 934
Java
  converting Java and SQL, 140
  installing, 868
  system tables, 1084
  user defined functions, 148
Java and SQL data type conversion
  about, 140
Java classes
  adding comments using the COMMENT statement, 538
  loaded in the database, 1165
  troubleshooting, 1165
Java data types
  converting from SQL, 141
  converting to SQL, 140
Java to SQL data type conversion
about, 140
Java VM
STOP JAVA statement, 1006
JavaScript Object Notation
CREATE SERVICE statement [HTTP web service], 466, 663
jConnect
CREATE DATABASE statement, 553
JCONNECT clause
ALTER DATABASE statement, 427
CREATE DATABASE statement, 553
JDBC
data type conversion, 140
Java to SQL data type conversion, 140
SQL to Java data type conversion, 141
upgrading database components, 426
join operators
compatibility with ASE, 20
joins
ANSI equivalency, 348
deleting rows based on joins, 741
FROM clause SQL syntax, 810
SQL Remote updates based on, 1035
updates based on, 1040
JSON
CREATE SERVICE statement [HTTP web service], 466, 663

K
Kerberos
adding comments using the COMMENT statement, 538
case sensitivity of principals, 838
granting Kerberos login, 838
granting privilege, 838
revoking KERBEROS LOGIN, 944
KERBEROS LOGIN
revoking using the REVOKE statement, 946
Kerberos logins
GRANT KERBEROS LOGIN statement, 838
KEY clause
ALTER DATABASE statement, 428
CREATE DECRYPTED DATABASE statement, 559
CREATE DECRYPTED FILE statement, 560
CREATE ENCRYPTED DATABASE statement, 565
CREATE ENCRYPTED FILE statement, 567
CREATE ENCRYPTED TABLE DATABASE statement, 565
DROP DATABASE statement, 754
RESTORE DATABASE statement, 938
START DATABASE statement, 994
KEY JOIN clause
FROM clause SQL syntax, 810
keys
CREATE SEQUENCE statement, 655
keywords
how to use SQL reserved words in syntax, 1
list of SQL reserved words, 1
sa_reserved_words system procedure, 1214
kind tests
supported by OPENXML operator, 15
labels
for statements, 422
statements, 827
LANGUAGE C_ESQL32 clause
CREATE FUNCTION statement [External call], 583
CREATE PROCEDURE statement [External call], 624
LANGUAGE C_ESQL64 clause
CREATE FUNCTION statement [External call], 583
CREATE PROCEDURE statement [External call], 624
LANGUAGE C_ODBC32 clause
CREATE FUNCTION statement [External call], 583
CREATE PROCEDURE statement [External call], 624
LANGUAGE C_ODBC64 clause
CREATE FUNCTION statement [External call], 583
CREATE PROCEDURE statement [External call], 624
LANGUAGE CLR clause
CREATE FUNCTION statement [External call], 583
CREATE PROCEDURE statement [External call], 583
CREATE PROCEDURE statement [External call], 624
language elements
comments, 85
constants, 6
expressions, 21
identifiers, 4
keywords, 1
named parameters, 87
operators, 9
search conditions, 40
special values, 65
strings, 6
syntax, 1
variables, 79
LANGUAGE JAVA clause
CREATE FUNCTION statement [External call], 584
CREATE PROCEDURE statement [External call], 625
LANGUAGE PERL clause
CREATE FUNCTION statement [External call], 583
CREATE PROCEDURE statement [External call], 625
LANGUAGE PHP clause
CREATE FUNCTION statement [External call], 584
CREATE PROCEDURE statement [external procedures], 625
large binary objects
getting from columns, 822
large databases
index storage, 601
LAST clause
FETCH statement , 802
LAST USER clause
CREATE TABLE statement, 690
LAST USER default
CREATE TABLE statement, 491, 697
LAST USER special value
syntax, 71
LAST_VALUE function
syntax, 279
lateral derived tables
FROM clause outer references, 813
LCASE function
syntax, 281
LDAP SERVER
adding comments using the COMMENT statement, 538
LDAP server configuration objects
SYSLDAPSERVER system view, 1369
LDAP servers
altering LDAP server configuration objects, 442
creating LDAP server configuration objects, 603
dropping LDAP server configuration objects, 760
obtaining status, 1147
validating LDAP server configuration objects, 1044
LDAP user authentication
altering LDAP server configuration objects, 442
creating LDAP server configuration objects, 603
sa_get_lapserv_status system procedure, 1147
ldap_auto_failback_period option
ALTER LOGIN POLICY statement, 446
ldap_failover_to_std option
ALTER LOGIN POLICY statement, 446
ldap_primary_server option
ALTER LOGIN POLICY statement, 446
ldap_refresh_dn option
ALTER LOGIN POLICY statement, 446
ldap_secondary_server option
ALTER LOGIN POLICY statement, 446
CREATE LOGIN POLICY statement, 608
leap years
about, 115
LEAVE statement
syntax, 871
LEFT function
syntax, 282
LEFT OUTER JOIN clause
FROM clause SQL syntax, 810
LEN function
syntax, 283
LENGTH clause
GET DESCRIPTOR statement [ESQL], 824
SET DESCRIPTOR statement [ESQL], 967
LENGTH function
syntax, 283
less than
comparison operator, 9
Index

less than or equal to
comparison operator, 9
LESSER function
syntax, 284
LIKE search condition
case-sensitivity, 49
collations, 49
compared to REGEXP and SIMILAR TO, 45
pattern length, 48
syntax, 47
LIKE, REGEXP, and SIMILAR TO search conditions
about, 45
LIMIT clause
ATTACH TRACING statement, 514
SELECT statement, 957
limiting the number of rows returned
about, 955
line interpretation, spatial data
planar type, 676
round-earth type, 676
LINEAR UNIT OF MEASURE clause
ALTER SPATIAL REFERENCE SYSTEM
statement, 478
CREATE SPATIAL REFERENCE SYSTEM
statement, 676
LIST function
syntax, 285
lists
LIST function syntax, 285
sa_split_list system procedure, 1248
storing as rows or arrays, 128
literals
about, 6
LivenessTimeout property
setting with sa_server_option, 1222
LOAD privilege
GRANT statement, 828
REVOKE statement, 946
LOAD privilege clause
GRANT statement, 830
LOAD STATISTICS statement
syntax, 873
LOAD TABLE statement
syntax, 873
using with database mirroring, 886
loading
bulk inserts, 873
cost models, 1167
fails if table has an immediate text index, 885
fails if table referenced by an immediate views, 885
LOAD TABLE statement, 873
loading data from a file on the client computer, 875
loading data from a file on the database server computer, 875
loading data from a specified value, 876
shapefiles using st_geometry_load_shapefile system procedure, 1327
loading data
multibyte character sets, 882
LOCAL DATABASE clause
ATTACH TRACING statement, 514
local temporary tables
creating, 735
creating using the CREATE LOCAL TEMPORARY TABLE statement, 606
local variables
definition, 79
syntax, 80
LOCATE function
syntax, 287
Location
HTTP_RESPONSE_HEADER function, 268
LOCATION clause
ALTER EXTERNAL ENVIRONMENT statement, 438
LOCK FEATURE statement
syntax, 889
LOCK TABLE statement
syntax, 890
locked accounts
determining cause, 1152
locked option
ALTER LOGIN POLICY statement, 446
CREATE LOGIN POLICY statement, 608
locking
blocks, 1106
features, 889
tables, 890
locks
displaying, 1168
log files
allocating space using ALTER DBSPACE, 432
analyzing the request log using
sa_get_request_profile, 1148
analyzing the request log using
  sa_get_request_times, 1149
determining available space, 1133
LOG function
  syntax, 289
LOG10 function
  syntax, 289
LogDiskSpace system event
  example, 240
logfile option
  ALTER MIRROR SERVER statement, 454
  CREATE MIRROR SERVER statement, 618
LogFreePercent event condition
  about, 238
LogFreeSpace event condition
  about, 238
logging
  deadlocks, 1222
  START LOGGING statement, 999
  STOP LOGGING statement, 1007
  updating columns without, 1054
logical operators
  syntax, 10
  three-valued logic, 63
login policies
  adding comments using the COMMENT statement, 538
  ALTER LOGIN POLICY statement, 445
  ALTER USER statement, 509
  CREATE LOGIN POLICY statement, 607
  CREATE USER statement, 721
  default login policy, 607
  DROP LOGIN POLICY statement, 761
  policy options, 607
LOGIN POLICY clause
  ALTER USER statement, 508
  CREATE USER statement, 721
logins
  assigning for remote servers, 578
  disabling connections to a server, 1222
  dropping for remote servers, 758
  obtaining status, 1152
LogSize event condition
  about, 238
LONG BINARY data type
  syntax, 126
LONG NAMES clause
  DESCRIBE statement, 745
LONG NVARCHAR data type
  describing, 91
  syntax, 91
LONG VARBIT data type
  syntax, 107
LONG VARCHAR data type
  syntax, 91
LOOP clause
  LOOP statement, 891
looping
  over cursors, 805
lowercase conversions
  about, 132
LOWER function
  syntax, 290
lowercase strings
  LCASE function, 281
  LOWER function, 290
LTRIM function
  syntax, 291
MANUAL REFRESH clause
  ALTER MATERIALIZED VIEW statement, 451
  CREATE MATERIALIZED VIEW statement, 612
manual views
  ALTER MATERIALIZED VIEW statement, 450
MAPI
  extended system procedures, 1086
  MAPI and SMTP system procedures, 1336
  return codes for MAPI and SMTP system procedures, 1086
  starting email sessions, 1340
  stopping email sessions, 1343
MAPI and SMTP system procedures
  return codes, 1086
MATCH clause
  ALTER TABLE statement, 492
  CREATE TABLE statement, 699
MATCH FULL clause
  CREATE TABLE statement, 699
MATCH SIMPLE clause
  CREATE TABLE statement, 699
Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.01501
match types
   referential integrity, 699
MATCH UNIQUE FULL clause
   CREATE TABLE statement, 699
MATCH UNIQUE SIMPLE clause
   CREATE TABLE statement, 699
MATERIALIZED VIEW clause
   TRUNCATE statement, 1019
   VALIDATE statement, 1045
MATERIALIZED VIEW OPTIMIZATION clause
   DELETE statement, 743
   EXCEPT statement, 790
   INSERT statement, 863
   INTERSECT statement, 870
   SELECT statement, 962
   UPDATE statement, 1040
MATERIALIZED VIEW OPTIMIZATION option
   MERGE statement, 897
materialized view properties
   RefreshType property, 1176
materialized views
   adding comments using the COMMENT statement, 538
   ALTER INDEX statement, 441
   ALTER MATERIALIZED VIEW statement, 450
   altering materialized views owned by someone else, 451
   altering using SQL, 450
   CREATE MATERIALIZED VIEW statement, 612
   creating using SQL, 612
   creation options stored in SYSMVOPTION, 1373
   decrypting using SQL, 450
   determining status, 1176
   disabling use in optimization using SQL, 450
   disabling using SQL, 450
   DROP MATERIALIZED VIEW statement, 762
   dropping using SQL, 762
   enabling use in optimization using SQL, 450
   enabling using SQL, 450
   encrypting using SQL, 450
   listing all materialized views in the database, 1176
   privileges required for immediate views, 451
   querying using index hints, 818
   querying using table hints, 816
   REFRESH MATERIALIZED VIEW statement, 925
   refreshing using SQL, 925
   setting isolation level for refreshing, 925
   testing eligibility for becoming an immediate view, 1174
   unloading using the UNLOAD statement, 1027
   validating indexes, 1045
materialized-view-name
   common element in SQL syntax, 421
materialized_view_optimization option
   overriding in a SELECT statement, 962
   setting for DELETE statements, 743
   setting for EXCEPT statement, 790
   setting for INSERT statements, 863
   setting for INTERSECT clauses, 870
   setting for INTERSECT statement, 870
   setting for UNION statement, 1026
   setting for UPDATE statements, 1040
mathematical expressions
   arithmetic operators, 11
MAX function
   syntax, 292
MAX WRITE clause
   syntax, 292
max_connections option
   ALTER LOGIN POLICY statement, 446
   CREATE LOGIN POLICY statement, 608
max_days_since_login option
   ALTER LOGIN POLICY statement, 446
   CREATE LOGIN POLICY statement, 608
max_disconnected_time option
   ALTER LOGIN POLICY statement, 446
   CREATE LOGIN POLICY statement, 608
max_failed_login_attempts option
   ALTER LOGIN POLICY statement, 446
   CREATE LOGIN POLICY statement, 608
max_non_dba_connections option
   ALTER LOGIN POLICY statement, 446
   CREATE LOGIN POLICY statement, 608
max_query_tasks option
   overriding in a MERGE statement, 897
   setting for DELETE statements, 743
   setting for EXCEPT statement, 790
   setting for INSERT statements, 863
   setting for INTERSECT statement, 870
   setting for UNION statement, 1026
   setting for UPDATE statements, 1040
max_retry_connect_time option
   ALTER LOGIN POLICY statement, 446
   CREATE LOGIN POLICY statement, 608
   SET MIRROR OPTION statement, 969
maximum
   date ranges, 115
MAXIMUM TERM LENGTH clause
ALTER TEXT CONFIGURATION statement, 501
MaxMultiProgrammingLevel property
setting with sa_server_option, 1222
MAXVALUE clause
ALTER SEQUENCE statement, 462
CREATE SEQUENCE statement, 656
MEDIAN function
syntax, 293
memory
allocating for descriptor areas, 424
MERGE clause
ALTER SYNCHRONIZATION PROFILE
statement, 482
MERGE statement
syntax, 892
merging
MERGE statement, 892
message control parameters
setting, 978
MESSAGE LOG clause
LOAD TABLE statement, 882
MESSAGE statement
syntax, 899
MessageCategoryLimit property
setting with sa_server_option, 1222
messages
creating, 614
displaying, 899
dropping remote types, 767
dropping using the DROP MESSAGE statement,
763
MESSAGE statement, 899
SQL Remote altering remote types, 459
SQL Remote creating remote types, 651
messages windows
printing messages in, 918
metacharacters
list of metacharacters used in regular expressions,
27
metaproperties
LOAD TABLE statement, 881
OPENXML operator, 13
METHODS clause
ALTER SERVICE statement [HTTP web service],
466
ALTER SERVICE statement [SOAP web service],
471
CREATE SERVICE statement [HTTP web service], 468, 665
CREATE SERVICE statement [SOAP web service], 475, 671
migrating
compatibility roles using ALTER ROLE, 460
migrating databases
sa_migrate system procedure, 1183
MIME base64
decoding data, 165
encoding data, 166
MIN function
syntax, 295
minimum
date ranges, 115
MINIMUM TERM LENGTH clause
ALTER TEXT CONFIGURATION statement, 501
MinMultiProgrammingLevel property
setting with sa_server_option, 1222
MINUTE function
syntax, 296
MINUTES function
syntax, 297
MINVALUE clause
ALTER SEQUENCE statement, 462
CREATE SEQUENCE statement, 656
MIRROR clause
CREATE DATABASE statement, 554
MIRROR ON clause
START DATABASE statement, 995
mirror servers
adding comments using the COMMENT statement,
538
ALTER MIRROR SERVER statement, 452
altering with the ALTER MIRROR SERVER
statement, 452
CREATE MIRROR SERVER statement, 615
creating with the CREATE MIRROR SERVER
statement, 615
defining with CREATE MIRROR SERVER
statement, 616
dropping with the DROP MIRROR SERVER
statement, 764
mirroring
dropping servers, 764
MirrorServerState property
DB_EXTENDED_PROPERTY function, 212
MirrorState property
DB_EXTENDED_PROPERTY function, 212
mixed storage format
ALTER SPATIAL REFERENCE SYSTEM statement, 479
CREATE SPATIAL REFERENCE SYSTEM statement, 679
MobiLink
ALTER PUBLICATION statement, 457
ALTER SYNCHRONIZATION PROFILE statement, 481
ALTER SYNCHRONIZATION SUBSCRIPTION statement, 482
ALTER SYNCHRONIZATION USER statement, 485
CREATE PUBLICATION statement, 647
CREATE SYNCHRONIZATION PROFILE statement, 685
CREATE SYNCHRONIZATION SUBSCRIPTION statement, 686
CREATE SYNCHRONIZATION USER statement, 688
DROP PUBLICATION statement, 766
DROP SYNCHRONIZATION PROFILE statement, 778
DROP SYNCHRONIZATION SUBSCRIPTION statement, 778
START SYNCHRONIZATION DELETE statement, 1001
START SYNCHRONIZATION SCHEMA change statement, 1002
STOP SYNCHRONIZATION DELETE statement, 1010
STOP SYNCHRONIZATION SCHEMA change statement, 1015
MobiLink users
ALTER SYNCHRONIZATION USER statement, 485
CREATE SYNCHRONIZATION USER statement, 688
DROP SYNCHRONIZATION USER statement, 779
MOD function
syntax, 299
money
money data types, 106
MONEY data type
syntax, 106
money data types
MONEY, 106
SMALLMONEY, 107
monitoring performance
execution time determination, 1149
MONTH function
syntax, 299
MONTHNAME function
syntax, 300
MONTHS function
syntax, 301
multi-row fetches
FETCH statement, 802
OPEN statement, 905
multi-row inserts
about, 794
multibyte character sets
loading data, 882
unloading data, 1029
multiple result sets
retrieving, 939
multiprogramming level
controlling automatic tuning, 1222
setting current value, 1222
setting maximum value, 1222
setting minimum value, 1222
viewing statistics, 1222
N
n-gram
ALTER TEXT CONFIGURATION statement, 502
NAME clause
GET DESCRIPTOR statement [ESQL], 824
named parameters
about, 87
allowed when calling procedures, 530
names
column names, 23
NAMESPACE clause
CREATE FUNCTION statement [web clients], 590
CREATE PROCEDURE statement [web clients], 635
native function calls
functions, 582
interfacing to native functions using CREATE FUNCTION statement, 580
interfacing to native procedures using CREATE PROCEDURE statement, 620
procedures, 624
native procedures
interfacing to native procedures using CREATE PROCEDURE statement, 620
NATURAL JOIN clause
FROM clause SQL syntax, 810
NCHAR COLLATION clause
collation tailoring, 554
CREATE DATABASE statement, 554
NCHAR data type
comparing with CHAR data type, 132
describing, 92
syntax, 92
use with the LIKE search condition, 50
use with the REGEXP search condition, 52
use with the SIMILAR TO search condition, 54
using DESCRIBE on an NCHAR column, 92
NCHAR function
syntax, 302
NcharCollation property
DB_EXTENDED_PROPERTY function, 212
NEAR clause
CONTAINS search condition, 55
nesting
user-defined transactions using the BEGIN TRANSACTION statement, 527
NEW clause
INSTALL EXTERNAL OBJECT statement, 866
INSTALL JAVA statement, 868
new line characters
in SQL strings, 8
NEW SUBSCRIBE BY clause
UPDATE statement, 1037
UPDATE statement [SQL Remote], 1034
NEWID function
syntax, 303
NEXT clause
FETCH statement, 801
NEXT_CONNECTION function
syntax, 304
NEXT_DATABASE function
syntax, 306
NEXT_HTTP_HEADER function
syntax, 306
NEXT_HTTP_RESPONSE_HEADER function
syntax, 308
NEXT_HTTP_VARIABLE function
syntax, 309
NEXT_SOAP_HEADER function
syntax, 310
NextScheduleTime property
DB_EXTENDED_PROPERTY function, 212
NO COPY clause
BACKUP statement, 519
NO INDEX clause
ALTER TABLE statement, 492
CREATE TABLE statement, 694
FROM clause, 818
NO RESULT SET clause
CREATE PROCEDURE statement, 642
CREATE PROCEDURE statement [External call], 623
CREATE PROCEDURE statement [T-SQL], 637
NO SCROLL clause
DECLARE CURSOR statement, 730
FOR statement, 805
NO SCROLL cursors
declaring, 729
NOLOCK table hint
FROM clause, 816
NONCLUSTERED clause
ALTER INDEX statement, 441
NOSTRIP clause
INPUT statement, 858
NOT
bitwise operators, 19
logical operators description, 10
three-valued logic, 63
NOT COMPRESSED clause
LOAD TABLE statement, 878
NOT DETERMINISTIC clause
CREATE FUNCTION statement [External call], 580
CREATE FUNCTION statement [external procedures], 581
CREATE FUNCTION statement [user defined], 595
NOT ENCRYPTED clause
ALTER MATERIALIZED VIEW statement, 451
LOAD TABLE statement, 879
not equal to
comparison operator, 9
not greater than
comparison operator, 9
not less than
  comparison operator, 9
NOT NULL clause
  ALTER TABLE statement , 488
  CREATE EXISTING TABLE statement, 576
  CREATE TABLE statement, 694
NOT TRANSACTIONAL clause
  CREATE LOCAL TEMPORARY TABLE statement, 606
  CREATE TABLE statement, 692
  DECLARE LOCAL TEMPORARY TABLE statement, 735
NOTFOUND clause
  WHENEVER statement [ESQL], 1049
NOTIFY TRACE EVENT statement
  syntax, 903
NOW function
  syntax, 311
NTEXT data type
  syntax, 93
NULL
  about, 71
  ASE compatibility, 73
  ISNULL function, 277
  NULL value, 71
  returned by functions if a NULL argument is specified, 143
  space allocated for NULL values, 72
  three-valued logic, 72
NULL clause
  ALTER TABLE statement , 488
  CASE statement, 532
  CASE statement [T-SQL], 534
  CREATE DOMAIN statement, 562
  CREATE TABLE statement, 694
NULL constants
  converting to NUMERIC, 138
  converting to string types, 138
NULL values
  domains, 562
NULLABLE clause
  GET DESCRIPTOR statement [ESQL], 824
NULLIF function
  about, 312
  using with CASE expressions, 25
  number
    common element in SQL syntax, 421
NUMBER function
  SQL Remote updates, 1035
    syntax, 313
  number of rows
    system views, 1399
NUMERIC data type
  comparing with dates and times, 134
  converting from DOUBLE, 140
  syntax, 103
numeric data types
  about, 97
  BIGINT, 97
  BIT, 98
  converting DOUBLE to NUMERIC, 140
  DECIMAL, 99
  DOUBLE, 100
  FLOAT, 101
  INTEGER, 102
  NUMERIC, 103
  REAL, 104
  SMALLINT, 105
  TINYINT, 105
numeric functions
  alphabetical list, 151
NVARCHAR data type
  describing, 93
  syntax, 93
  using DESCRIBE on an NVARCHAR column, 93
O
  object-level privileges
    granting using the GRANT statement, 828
    revoking using the REVOKE statement, 946
OBJECT_ID function
  syntax, 154
OBJECT_NAME function
  syntax, 154
objects
  listing user privileges, 1276, 1292
ODBC
  declaring static cursors, 729
OFFSET clause
  GET DATA statement , 823
  SELECT statement, 957
OLAP
  CUBE operation, 848
  GROUP BY clause, 847
  GROUPING function, 255
GROUPING SETS operation, 847
ROLLUP operation, 847
WINDOW clause, 1051
OLAP functions
AVG function, 164
COUNT function, 192
COUNT_BIG function, 193
COVAR_POP function, 195
CUME_DIST function, 200
DENSE_RANK function, 222
MAX function, 292
MEDIAN function, 293
MIN function, 295
PERCENT_RANK function, 316
RANK function, 326
REGR_AVGX function, 331
REGR_AGGY function, 332
REGR_COUNT function, 334
REGR_INTERCEPT function, 335
REGR_R2 function, 337
REGR_SLOPE function, 338
REGR_SXX function, 339
REGR_SXY function, 341
ROW_NUMBER function, 353
STDDEV function, 372
STDDEV_POP function, 372
STDDEV_SAMP function, 373
SUM function, 381
VAR_POP function, 402
VAR_SAMP function, 404
OLD KEY clause
CREATE ENCRYPTED FILE statement, 567
CREATE ENCRYPTED TABLE DATABASE statement, 565
OLD SUBSCRIBE BY clause
UPDATE statement, 1037
UPDATE statement [SQL Remote], 1034
ON clause
ALTER STATISTICS statement, 480
ALTER TRIGGER statement, 507
CREATE EVENT statement, 573
CREATE INDEX statement, 599
CREATE TEXT INDEX statement, 711
CREATE TRIGGER [Transact-SQL] statement, 720
DROP STATISTICS statement, 775
DROP TEXT INDEX statement, 782
MERGE statement, 895
TRUNCATE TEXT INDEX statement, 1021
VALIDATE statement, 1045
ON COMMIT clause
CREATE LOCAL TEMPORARY TABLE statement, 606
CREATE TABLE statement, 692
DECLARE LOCAL TEMPORARY TABLE statement, 735
ON DATABASE clause
STOP DATABASE statement, 1004
ON EXCEPTION RESUME clause
about, 642
CREATE FUNCTION statement [user defined], 595
CREATE PROCEDURE statement, 642
ON EXISTING clause
INSERT statement, 862
ON EXISTING ERROR clause
BACKUP statement, 518
behavior with DEFAULT columns, 863
ON EXISTING SKIP clause
behavior with DEFAULT columns, 863
ON phrase
search conditions, 40
on_tsql_error option
and ON EXCEPTION RESUME clause, 642
OPEN statement
embedded SQL syntax, 904
syntax, 904
WITH HOLD syntax, 904
opening cursors
OPEN statement, 904
OPENSTRING clause
example, 821
FROM clause, 814
OPENXML operator
list of supported metaproperties, 13
supported kind tests, 15
syntax, 12
operating systems
executing commands, 1017
operational servers
ALTER MIRROR SERVER statement, 452
CREATE MIRROR SERVER statement, 615
operator precedence
full text search, 58
syntax, 20
operators
about, 9
arithmetic operators, 11
array operators, 18
bitwise operators, 19
comparison operators, 9
logical operators description, 10
OPENXML operator, 12
precedence, 20
string operators, 11
optimization
avoiding using FORCE NO OPTIMIZATION clause, 962
defining existing tables and, 577
forcing using FORCE OPTIMIZATION option, 962
optimization_goal option
overriding in a MERGE statement, 897
setting for DELETE statements, 743
setting for EXCEPT statement, 790
setting for INSERT statements, 863
setting for INTERSECT statement, 870
setting for UNION statement, 1026
setting for UPDATE statements, 1040
optimization_level option
overriding in a MERGE statement, 897
setting for DELETE statements, 743
setting for EXCEPT statement, 790
setting for INSERT statements, 863
setting for INTERSECT statement, 870
setting for UNION statement, 1026
setting for UPDATE statements, 1040
optimization_workload option
overriding in a MERGE statement, 897
setting for DELETE statements, 743
setting for EXCEPT statement, 790
setting for INSERT statements, 863
setting for INTERSECT statement, 870
setting for UNION statement, 1026
setting for UPDATE statements, 1040
optimizer
CREATE STATISTICS statement, 682
explicit selectivity estimates, 64
optimizer plans
getting text specification, 799
optimizer statistics
dropping using the DROP STATISTICS statement, 775
optimizer tables
about, 1069
OPTIOIN clause
CREATE SYNCHRONIZATION
SUBSCRIPTION statement [MobiLink], 687
CREATE SYNCHRONIZATION USER, 689
DELETE statement, 743
EXCEPT statement, 790
INSERT statement, 863
INTERSECT statement, 870
MERGE statement, 897
SELECT statement, 962
UNION statement, 1026
UPDATE statement, 1040
options
getting values, 825
initial settings for sp_login_environment system procedure, 1289
initial settings for sp_tsql_environment system procedure, 1320
overriding, 1222
quoted_identifier and T-SQL compatibility, 39
SET OPTION statement, 972
setting in Interactive SQL, 975
setting in Transact-SQL, 987
setting remote, 978
setting with sp_tsql_environment system procedure, 1320
SYSOPTIONS consolidated view, 1428
system views, 1375
SYSUSEROPTIONS consolidated view, 1438
options watch list
configuring with sa_server_option, 1222
OptionWatchAction property
setting with sa_server_option, 1222
OptionWatchList property
setting with sa_server_option, 1222
OR
bitwise operators, 19
logical operators description, 10
three-valued logic, 63
OR REPLACE clause
CREATE FUNCTION statement [External call], 580
CREATE FUNCTION statement [external procedures], 581
CREATE FUNCTION statement [user defined], 595
CREATE FUNCTION statement [Web service], 586
CREATE MIRROR SERVER statement, 616
CREATE PROCEDURE statement, 639
CREATE PROCEDURE statement [External call], 620
CREATE PROCEDURE statement [T-SQL], 637
CREATE PROCEDURE statement [web clients], 629
CREATE SEQUENCE statement, 656
CREATE SPATIAL REFERENCE SYSTEM statement, 673
CREATE SYNCHRONIZATION PROFILE statement, 685
CREATE TRIGGER statement, 714
CREATE VARIABLE statement, 722
CREATE VIEW statement, 724

Oracle databases
migrating to SQL Anywhere using sa_migrate system procedure, 1184

ORDER BY clause
about, 960
DELETE statement, 742
EXCEPT statement, 789
INTERSECT statement, 870
SELECT statement, 955
UPDATE statement, 1040
UPDATE statement [SQL Remote], 1034
WINDOW clause, 1052

ORDER clause
CREATE TRIGGER statement, 715
LOAD TABLE statement, 882
UNLOAD statement, 1029

order of operations
SQL operator precedence, 20

order of SQL clauses
syntax conventions, 422

ordinal date
ISO 8601, 110

ORGANIZATION clause
ALTER SPATIAL REFERENCE SYSTEM statement, 478
CREATE SPATIAL REFERENCE SYSTEM statement, 675

original storage format
ALTER SPATIAL REFERENCE SYSTEM statement, 479

CREATE SPATIAL REFERENCE SYSTEM statement, 679
OUT clause
CREATE PROCEDURE statement, 640
out of disk space
creating events using the CREATE EVENT statement, 570

OUTER APPLY clause
FROM clause, 815
outer references
FROM clause, 813
lateral derived tables, 813

OUTPUT clause
CREATE PROCEDURE statement [T-SQL], 637
DESCRIBE statement, 745

OUTPUT statement
text, 907
output_log_send_limit
SQL Remote syntax, 978
output_log_send_now
SQL Remote syntax, 978
output_log_send_on_error
SQL Remote syntax, 978

overflow errors
AVG function, 164
SUM function, 381

owner
common element in SQL syntax, 421

OWNER.TABLE.COLUMN clause
DESCRIBE statement [ESQL], 744

OwnerName property
sa_materialized_view_info system procedure, 1176

P

text
installing Java classes, 868
removing Java classes, 934

PAGE SIZE clause
CREATE DATABASE statement, 553

page sizes
creating databases, 553

page usage
tables, 1254

page_timeout option
SET MIRROR OPTION statement, 969

parallel backups
BACKUP statement, 516
Index

parameterized views  
about, 725

parameters  
Interactive SQL script files, 912  
named, 87

PARAMETERS statement  
Interactive SQL syntax, 912

PARTITION BY clause  
WINDOW clause, 1052

partner servers  
ALTER MIRROR SERVER statement, 452  
changing with ALTER MIRROR SERVER statement, 453  
CREATE MIRROR SERVER statement, 615  
defining with CREATE MIRROR SERVER statement, 616

PASSTHROUGH FOR clause  
PASSTHROUGH statement [SQL Remote], 913

PASSTHROUGH FOR SUBSCRIPTION clause  
PASSTHROUGH statement [SQL Remote], 913

passthrough mode  
FORWARD TO statement, 808  
PASSTHROUGH statement [SQL Remote], 913  
starting, 913  
stopping, 913

PASSTHROUGH ONLY clause  
PASSTHROUGH statement [SQL Remote], 913

PASSTHROUGH statement  
SQL Remote syntax, 913

PASSTHROUGH STOP clause  
PASSTHROUGH statement [SQL Remote], 913

password_expiry_on_next_login option  
ALTER LOGIN POLICY statement, 446  
CREATE LOGIN POLICY statement, 608

password_grace_time option  
ALTER LOGIN POLICY statement, 446  
CREATE LOGIN POLICY statement, 608

password_life_time option  
ALTER LOGIN POLICY statement, 446  
CREATE LOGIN POLICY statement, 608

passwords  
character set conversion, CREATE USER statement, 722  
character set conversion, GRANT CONNECT statement, 833  
maximum length, ALTER USER statement, 510  
maximum length, CREATE USER statement, 722  
Sa_verify_password system procedure, 1267

passwords, character set conversion on  
ALTER USER statement, 510

PATINDEX function  
syntax, 314

pattern length  
LIKE search condition, 48

pattern matching  
case-sensitivity, 49  
collations, 49  
LIKE search condition, 47  
PATINDEX function, 314  
SIMILAR TO search condition, 53

PCTFREE clause  
CREATE LOCAL TEMPORARY TABLE statement, 606  
CREATE TABLE statement, 700  
DECLARE LOCAL TEMPORARY TABLE statement, 735  
LOAD TABLE, 882

PCTFREE setting  
ALTER TABLE statement, 486  
CREATE LOCAL TEMPORARY TABLE syntax, 606  
CREATE TABLE statement, 690  
DECLARE LOCAL TEMPORARY TABLE syntax, 735  
LOAD TABLE syntax, 873

percent sign  
comment indicator, 86

PERCENT_RANK function  
syntax, 316

performance  
compression statistics, 1104  
preallocating space, 433  
recalibrating the database server, 426  
recalibrating the I/O cost model, 429  
SQL Remote updates, 1036

PERL clause  
INSTALL EXTERNAL OBJECT statement, 866

permissions  
SYSCOLAUTH view, 1425

PHP clause  
INSTALL EXTERNAL OBJECT statement, 866
physical indexes
recorded in SYSPHYSIDX system view, 1376

PI function
syntax, 318

PLAN function
syntax, 318

planar model
ALTER SPATIAL REFERENCE SYSTEM
statement, 479
CREATE SPATIAL REFERENCE SYSTEM
statement, 676

plans
and cursors, 318
example of saving a plan to a file, 253
EXPLANATION function, 244
getting text specification, 799
GRAPHICAL_PLAN function, 252
PLAN function, 318
TRACED_PLAN function, 390

policy options
altering using ALTER LOGIN POLICY statement, 445
altering using ALTER USER statement, 508

POLYGON FORMAT clause
ALTER SPATIAL REFERENCE SYSTEM
statement, 479
CREATE SPATIAL REFERENCE SYSTEM
statement, 678

polygon formats
Clockwise, 679
CounterClockwise, 679
EvenOdd, 679

polygon orientation
Clockwise, 679
CounterClockwise, 679
EvenOdd, 679

pooling
enabling connection pooling, 990

population covariance
about, 195

population variance
about, 402

position locks
sa_locks system procedure, 1170

positioned DELETE statement
syntax, 739

POWER function
syntax, 319

precedence
SQL operator precedence, 20

PRECEDING clause
WINDOW clause, 1052

PRECISION clause
GET DESCRIPTOR statement [ESQL], 824
SET DESCRIPTOR statement [ESQL], 967

predicates
about, 40
ALL search condition, 42
ANY search condition, 43
BETWEEN search condition, 45
comparison operators, 9
CONTAINS search condition, 45
EXISTS search condition, 62
external selectivity estimates, 64
IN search condition, 54
IS DISTINCT FROM, 44
IS NOT DISTINCT FROM, 44
IS NOT NULL search condition, 62
IS NULL search condition, 62
IS TRUE or FALSE search conditions, 63
IS UNKNOWN search condition, 63
LIKE search condition, 47
REGEXP search condition, 51
SIMILAR TO search condition, 53
SOME search condition, 43
SQL subqueries in, 42
taxt, 40	hree-valued logic, 63

preferred option
ALTER MIRROR SERVER statement, 454
CREATE MIRROR SERVER statement, 618

PREFILTER EXTERNAL NAME clause
ALTER TEXT CONFIGURATION statement, 502

prefilters
ALTER TEXT CONFIGURATION statement, 502
dropping using ALTER TEXT CONFIGURATION statement, 502

PREFIX clause
ALTER TABLE statement, 492
CREATE TABLE statement, 693

prefix searches
CONTAINS search condition, 55

PREPARE statement
embedded SQL syntax, 915

PREPARE TO COMMIT statement
syntax, 917
prepared statements
  dropping using the DROP STATEMENT statement, 775
  executing, 794
preparing
  for two-phase commit, 917
PRESERVE clause
  CREATE LOCAL TEMPORARY TABLE statement, 606
  DECLARE LOCAL TEMPORARY TABLE statement, 735
PRIMARY KEY clause
  ALTER INDEX statement, 441
  ALTER TABLE statement, 491
  CREATE TABLE statement, 697
  REORGANIZE TABLE statement, 935
  VALIDATE statement, 1045
PRIMARY KEY constraint clause
  CREATE TABLE statement, 697
primary key locks
  sa_locks system procedure, 1170
primary keys
  ALTER INDEX statement, 441
  clustering using the ALTER INDEX statement, 441
  CREATE SEQUENCE statement, 655
  generating unique values, 303
  generating unique values using UUIDs, 303
  integrity constraints in CREATE TABLE statement, 697
  order of columns in CREATE TABLE statement, 697
  remote tables and the sp_remote_exported_keys system procedure, 1300
  remote tables and the sp_remote_imported_keys system procedure, 1302
  renaming using the ALTER INDEX statement, 441
  UUIDs and GUIDs, 303
primary servers
  defining with CREATE MIRROR SERVER statement, 616
primary tables
  system views, 1361
PRINT statement
  Transact-SQL syntax, 918
printing
  messages in the messages window, 918
PRIOR clause
  FETCH statement, 801
privileges
  dbspaces, 836
  dropping for a table, 497
  generating reports, 1276, 1292
  granting CONSOLIDATE, 834
  granting object-level privileges using the GRANT statement, 828
  granting PUBLISH, 839
  granting REMOTE, 844
  granting system privileges using the GRANT statement, 828
  procedures, 836
  related system procedures, 1086
  revoking, 944
  revoking for sequences, 946
  revoking object-level privileges using the REVOKE statement, 946
  revoking system privileges using the REVOKE statement, 946
  sequences, 846
  sp_auth_sys_role_info system procedure, 1269
  sp_has_role system procedure, 1284
  sp_proc_priv system procedure, 1297
  sp_sys_priv_role_info system procedure, 1311
  SQL Remote REVOKE REMOTE statement, 944
  SQL Remote revoking, 948, 949
  SQL Remote revoking CONSOLIDATE, 942
  SQL Remote revoking PUBLISH, 943
  SYSCOLPERM system view, 1351
  SYSTABAUTH consolidated view, 1436
  SYSTABLEPERM system view, 1404
procedure
  listing privileges using sp_proc_priv, 1297
procedure calls
  invoking using the CALL statement, 530
PROCEDURE clause
  ALTER DATABASE statement, 427
  CREATE PUBLICATION statement [MobiLink][SQL Remote], 647
  DESCRIBE statement, 748
procedure parameters
  listing in Interactive SQL, 747
procedure profiling
  disabling from Interactive SQL, 1222
  enabling from Interactive SQL, 1222
  in Interactive SQL, 1222
  sa_procedure_profile system procedure, 1204
summary of procedures, 1206
viewing in Interactive SQL, 1206
ProcedureProfiling property
  setting with sa_server_option, 1222
procedures
  adding comments using the COMMENT statement, 538
  alphabetical list, 1092
  alphabetical list of system procedures, 1092
  altering using the ALTER PROCEDURE statement, 456
  compared to functions, 149
  CREATE PROCEDURE statement, 637
  creating in Transact-SQL, 637
  creating SQL stored procedures, 639
  creating web services, 628
  dropping using the DROP PROCEDURE statement, 765
  executing in dynamic SQL, 791
  executing stored in Transact-SQL, 796
  exiting, 940
  extended list, 1086
  GRANT EXECUTE statement, 836
  interfacing to native procedures using CREATE PROCEDURE statement, 620
  invoking using the CALL statement, 530
  named parameters, 87
  named parameters allowed when calling, 530
  native function calls, 624
  RAISERROR statement, 920
  replacing external procedure interface, 622
  replacing user-defined procedures, 640
  replacing web services, 629
  replicating using the ALTER PROCEDURE statement, 456
  resuming execution of, 939
  returning values from, 940
  revoking EXECUTE using the REVOKE statement, 946
  selecting from, 812
  system, 1085
  Transact-SQL list, 1089
  variable result sets, CREATE PROCEDURE statement, 641
  variable result sets, CREATE PROCEDURE statement [External call], 622
  variable result sets, DESCRIBE statement [ESQL], 746
variable result sets, PREPARE statement, 916
ProcessorAffinity property
  setting with sa_server_option, 1222
product name
  retrieving, 1333
ProfileFilterConn property
  setting with sa_server_option, 1222
ProfileFilterUser property
  setting with sa_server_option, 1222
promotion_time option
  SET MIRROR OPTION statement, 969
PROMPT clause
  INPUT statement, 858
properties
  CONNECTIONPROPERTY function, 186
  DB_PROPERTY function, 218
  PROPERTY function, 320
Properties property
  DB_EXTENDEDPROPERTY function, 212
PROPERTY function
  syntax, 320
PROPERTY_DESCRIPTION function
  syntax, 322
PROPERTY_NAME function
  syntax, 322
PROPERTY_NUMBER function
  syntax, 323
proximity searches
  CONTAINS search condition, 55
PROXY clause
  CREATE FUNCTION statement [web clients], 590
  CREATE PROCEDURE statement [web clients], 632
proxy procedures
  creating web services, 628
proxy tables
  CREATE TABLE statement, 692
  creating using the CREATE EXISTING TABLE statement, 576
PUBLIC clause
  SET OPTION statement, 972
PUBLIC system role
  granting using the GRANT statement, 828
  revoking using the REVOKE statement, 945
PUBLICATION clause
  UPDATE statement, 1037
  UPDATE statement [SQL Remote], 1034
publications
   ALTER PUBLICATION statement, 457
   CREATE PUBLICATION statement, 647
   DROP PUBLICATION statement, 766
   SQL Remote UPDATE statement, 1036
   UPDATE statement, 1040
PUBLISH privilege
   granting, 839
   SQL Remote revoking, 943
publisher
   address, 767
   GRANT PUBLISH statement, 839
   remote, 844
   SQL Remote address, 651
   SQL Remote addresses, 459
PunctuationSensitivity property
   DB_EXTENDED_PROPERTY function, 212
PURGE clause
   FETCH statement, 802
PUT statement
   embedded SQL syntax, 919
putting
   rows into cursors, 919
Q
quantifiers
   expression quantifiers, 27
QUARTER function
   syntax, 324
query block
   common element in SQL syntax, 421
query-block
   common element in SQL syntax, 421
query-expression
   common element in SQL syntax, 421
QUIT statement
   Interactive SQL syntax, 797
quitting
   Interactive SQL, 797
QuittingTime property
   setting with sa_server_option, 1222
quotation marks
   compatibility with ASE, 39
   database objects, 4
   single vs. double, 39
   SQL identifiers, 4
QUOTE clause
   LOAD TABLE statement, 882
   OUTPUT statement, 910
   UNLOAD statement, 1029
quoted_identifier option
   setting with Transact-SQL SET statement, 987
   T-SQL expression compatibility, 39
QUOTES clause
   LOAD TABLE statement, 882
   UNLOAD statement, 1029
R
R-squared
   regression lines, 337
RADIANS function
   syntax, 325
RAISERROR clause
   MERGE statement, 896
   RAISERROR statement
      syntax, 920
raising
   RAISERROR statement, 920
RAND function
   syntax, 325
   random numbers
   RAND function, 325
range
   date type, 115
RANGE clause
   WINDOW clause, 1052
RANK function
   syntax, 326
   ranking functions
      alphabetical list, 145
      CUME_DIST function, 200
      DENSE_RANK function, 222
      PERCENT_RANK function, 316
      RANK function, 326
rate properties
   incrementing values, 1263
   setting values, 1265
RAW
   CREATE SERVICE statement [HTTP web service], 466, 663
raw properties
   incrementing values, 1263
   setting values, 1265
re-describing cursors
CREATE PROCEDURE statement, 641
CREATE PROCEDURE statement [External call], 623
read committed
   FROM clause, 816
READ COMMITTED clause
   REFRESH MATERIALIZED VIEW statement, 925
read only
   backups that do not require recovery, 520
   defining read only mirror server, 616
   locking tables, 890
READ ONLY clause
   CREATE SERVER statement, 659
read only scale out
   ALTER MIRROR SERVER statement, 452
   CREATE MIRROR SERVER statement, 615
   dropping servers, 764
   SET MIRROR OPTION statement, 968
READ statement
   syntax, 922
read uncommitted
   FROM clause, 816
READ UNCOMMITTED clause
   REFRESH MATERIALIZED VIEW statement, 925
READ_CLIENT_FILE function
   syntax, 328
READCOMMITTED table hint
   FROM clause, 816
reading
   text and image values from the database, 924
   reading files
      using xp_read_file, 1334
   reading SQL statements from files
      about, 922
READONLY clause
   CREATE SERVER statement, 659
READPAST table hint
   FROM clause, 816
ReadPK locks
   sa_locks system procedure, 1170
READTEXT statement
   Transact-SQL syntax, 924
READUNCOMMITTED table hint
   FROM clause, 816
REAL data type
   comparing with dates and times, 134
syntax, 104
REBUILD clause
   ALTER INDEX statement, 441
RECOMPILE clause
   ALTER FUNCTION statement, 439
   ALTER PROCEDURE statement, 456
   ALTER VIEW statement, 512
RECOVER clause
   BACKUP statement, 519
REFERENCES clause
   ALTER TABLE statement, 492
   CREATE TABLE statement, 699
REFERENCES privilege
   GRANT statement, 828
   REVOKE statement, 946
REFERENCES privilege clause
   GRANT statement, 830
REFERENCING clause
   CREATE TRIGGER statement, 715
referential integrity
   FROM clause, 812
   match clause in CREATE TABLE statement, 699
Referer
   HTTP_HEADER function, 266
REFRESH clause
   ALTER TEXT INDEX statement, 505
   CREATE TEXT INDEX statement, 711
REFRESH DN clause
   ALTER USER statement, 509
REFRESH MATERIALIZED VIEW statement
   syntax, 925
REFRESH TEXT INDEX statement
   syntax, 928
REFRESH TRACING LEVEL statement
   diagnostic tracing, 930
   syntax, 930
REFRESH TRACING LEVELS statement
   populating sa_diagnostic_tracing_level table, 1081
refresh types
   RefreshType property, 1176
refreshing
   text index REFRESH TEXT INDEX, 928
   refreshing materialized views
      REFRESH clause, ALTER MATERIALIZED VIEW statement, 451
      REFRESH MATERIALIZED VIEWS statement, 925
   RefreshType property
Index

sa_materialized_view_info system procedure, 1176
REGEXP search condition
  compared to LIKE and SIMILAR TO, 45
database collation and matching, 52
  matching sub-character classes, 51
  regular expressions, 27
  syntax, 51
REGEXP_SUBSTR function
  database collation and matching, 330
  regular expressions, 27
  syntax, 329
REGR_AVGX function
  syntax, 331
REGR_AVGY function
  syntax, 332
REGR_COUNT function
  syntax, 334
REGR_INTERCEPT function
  syntax, 335
REGR_R2 function
  syntax, 337
REGR_SLOPE function
  syntax, 338
REGR_SXX function
  syntax, 339
REGR_SXY function
  syntax, 341
REGR_SYY function
  syntax, 342
regression functions
  REGR_AVGX function, 331
  REGR_AVGY function, 332
  REGR_COUNT function, 334
  REGR_INTERCEPT function, 335
  REGR_R2 function, 337
  REGR_SLOPE function, 338
  REGR_SXX function, 339
  REGR_SXY function, 341
  REGR_SYY function, 342
regular expressions
  about, 26
  assertion examples, 34
  database collations and matching, 46
  examples, 36
  list of assertions, 34
  metacharacters, 27
  REGEXP search condition, 27
  REGEXP syntax, 51
  REGEXP_SUBSTR function, 329
  SIMILAR TO search condition, 53
  special character classes, 30
  supported quantifiers, 27
  syntax, 27
  wildcard, grouping, and sets, 27
relationships
  system views, 1361
RELATIVE clause
  FETCH statement, 801
relative paths
  INPUT statement, 854
  READ statement, 922
RELEASE SAVEPOINT statement
  syntax, 931
releasing
  savepoints, 931
REMAINDER function
  syntax, 344
RememberLastPlan property
  setting with sa_server_option, 1222
RememberLastStatement property
  setting with sa_server_option, 1222
REMOTE clause
  CREATE EVENT statement, 570
remote data access
  disconnecting, 464
  FORWARD TO statement, 808
REMOTE DBA authority (deprecated)
  revoke, use REVOKE ROLE
    SYS_RUN_REPLICATION_ROLE instead, 945
remote functions
  AT clause, 596
REMOTE LOGIN clause
  CREATE EXTERNLOGIN statement, 579
remote message types
  dropping, 767
  SQL Remote altering, 459
  SQL Remote creating, 651
remote options
  SET REMOTE OPTION statement [SQL Remote], 978
REMOTE privilege
  SQL REMOTE granting, 844
  SQL Remote revoking, 944
remote procedures
  AT clause, 643
  CREATE PROCEDURE statement, 639
creating in Transact-SQL, 637
listing using sp_remote_procedures system procedure, 1307
RESULT clause requirement, 643
remote replication privileges
SQL Remote revoking, 948, 949
REMOTE RESET statement
SQL Remote syntax, 932
remote servers
altering attributes using the ALTER SERVER statement, 463
assigning logins for, 578
capabilities, determining using sp_servercaps, 1310
capabilities, SYSCAPABILITY system view, 1349
CREATE SERVER statement, 657
CREATE TABLE statement, 690
disconnecting, 464
dropping logins for remote servers, 758
dropping using the DROP SERVER statement, 772
sending SQL statements to, 808
SYSCAPABILITYNAME system view, 1349
remote stored procedures
parameters, 1303
result sets, 1303
remote tables
columns, 1298
CREATE TABLE statement, 692
foreign keys and the sp_remote_exported_keys system procedure, 1300
foreign keys and the sp_remote_imported_keys system procedure, 1302
listing using sp_remote_tables system procedure, 1308
primary keys and the sp_remote_exported_keys system procedure, 1300
primary keys and the sp_remote_imported_keys system procedure, 1302
remote users
SQL Remote REVOKE REMOTE statement, 944
remoteoption view
about, 1383
remoteoptiontype view
about, 1383
REMOVE EXTERNAL OBJECT statement
syntax, 933
REMOVE JAVA statement
syntax, 934
removing
Java classes, 934
privileges, 944
RENAME clause
ALTER DBSPACE statement, 433
ALTER DOMAIN statement, 435
ALTER INDEX statement, 441
ALTER SPATIAL REFERENCE SYSTEM statement, 477
ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 483
ALTER TABLE statement, 496
ALTER TEXT INDEX statement, 505
RESTORE DATABASE statement, 938
renaming
columns, 496
columns using the ALTER TABLE statement, 486
constraints, 496
tables, 496
tables using the ALTER TABLE statement, 486
REORGANIZE TABLE statement
syntax, 935
reorganizing tables
REORGANIZE TABLE, 935
REPEAT function
syntax, 345
REPEATABLE READ clause
REFRESH MATERIALIZED VIEW statement, 925
repeatable reads
FROM clause, 816
REPEATABLEREAD table hint
FROM clause, 816
REPLACE function
syntax, 345
replacing objects
sa_make_object, 1172
REPLICATE function
syntax, 347
replication
ALTER TABLE statement, 486
Replication roles
SYS_REPLICATION_ADMIN_ROLE system role, 840
SYS_RUN_REPLICATION_ROLE system role, 842
request log
processing with sa_get_request_times profile procedure, 1222
request logging
analyzing the request log with
sa_get_request_profile, 1148
analyzing the request log with
sa_get_request_times, 1149
enabling from Interactive SQL, 1222
request timing
sa_performance_diagnostics system procedure, 1196
RequestFilterConn property
setting with sa_server_option, 1222
RequestFilterDB property
setting with sa_server_option, 1222
RequestLogFile property
client statement caching, 1222
setting with sa_server_option, 1222
RequestLogging property
setting with sa_server_option, 1222
RequestLogMaxSize property
setting with sa_server_option, 1222
RequestLogNumFiles property
setting with sa_server_option, 1222
requests
obtaining timing information, 1196
RequestTiming property
setting with sa_server_option, 1222
reserved words
how to use in syntax, 1
list of SQL keywords, 1
sa_reserved_words system procedure, 1214
SQL Anywhere Server, 1
using as identifiers, 39
RESET LOGIN POLICY clause
ALTER USER statement, 509
RESIGNAL statement
syntax, 936
resignaling
exceptions, 936
RESOLVE clause
CREATE TRIGGER statement, 713
resolving conflicts
CONFLICT function for SQL Remote, 182
RESTART clause
ALTER DATABASE statement, 427
RESTART WITH clause
ALTER SEQUENCE statement, 462
RESTORE DATABASE statement
syntax, 937
RESTORE DEFAULT CALIBRATION clause
ALTER DATABASE statement, 428
restoring
databases from archives, 937
RESULT clause
CREATE PROCEDURE statement, 641
CREATE PROCEDURE statement [External call], 622
result sets
resuming execution of procedures, 939
retrieving multiple result sets, 939
selecting from stored procedures, 812
unloading using the UNLOAD statement, 1027
variable, CREATE PROCEDURE statement, 641
variable, CREATE PROCEDURE statement [External call], 622
variable, DESCRIBE statement [ESQL], 746
variable, PREPARE statement, 916
RESUME statement
not supported in Interactive SQL, 940
syntax, 939
resuming
execution of procedures, 939
retrieving
long column names, 745
multiple result sets, 939
retrieving dates and times from the database
about, 114
return codes
EXIT statement [Interactive SQL], 797
MAPI and SMTP system procedures, 1086
RETURN statement
syntax, 940
RETURNED_LENGTH clause
GET DESCRIPTOR statement [ESQL], 824
returning
values from procedures, 940
RETURNS clause
CREATE FUNCTION statement, 594
CREATE FUNCTION statement [External call], 580
CREATE FUNCTION statement [web clients], 587
REVERSE function
syntax, 347
REVOKE CONSOLIDATE statement
SQL Remote syntax, 942
REVOKE PUBLISH statement
SQL Remote syntax, 943
REVOKE REMOTE DBA statement
  use REVOKE ROLE
  SYS_RUN_REPLICATION_ROLE instead, 948
REVOKE REMOTE DBA statement (deprecated)
  use REVOKE ROLE
  SYS_RUN_REPLICATION_ROLE instead, 945
REVOKE REMOTE statement
  SQL Remote syntax, 944
REVOKE ROLE statement
  syntax, 944
REVOKE ROLE
SYS_REPLICATION_ADMIN_ROLE statement
  MobiLink syntax, 948
  SQL Remote syntax, 948
REVOKE ROLE SYS_RUN_REPLICATION_ROLE statement
  MobiLink syntax, 949
  SQL Remote syntax, 949
REVOKE statement
  syntax, 944
revoking
  REVOKE statement, 944
SQL Remote CONSOLIDATE privilege, 942
SQL Remote PUBLISH privilege, 943
SQL Remote REMOTE privilege, 944
SQL Remote remote replication privileges, 948, 949
revoking consolidate privileges
  SQL Remote CONSOLIDATE privileges, 942
revoking consolidated privileges
  REVOKE statement, 944
REWRITE function
  syntax, 348
RI constraints
  adding, deleting, or altering using the ALTER TABLE statement, 486
  ALTER TABLE statement, 690
  not renamed when underlying index is renamed, 441
  renaming using ALTER TABLE statement, 496
RIGHT function
  syntax, 350
RIGHT OUTER JOIN clause
  FROM clause SQL syntax, 810
role names
  foreign keys in CREATE TABLE statement, 698
role-name
  common element in SQL syntax, 421
roles
  altering (other than renaming) using the ALTER ROLE statement, 652
  converting user-extended roles back to users, 769
  creating user-extended roles, 652
  creating using CREATE ROLE statement, 652
  determining underlying system privileges, 1276
  dropping using the DROP ROLE statement, 769
  granting compatibility roles using the GRANT statement, 828
  granting system roles using the GRANT statement, 828
  granting user-defined roles using the GRANT statement, 828
  migrating compatibility roles using ALTER ROLE, 460
  related system procedures, 1086
  revoking compatibility roles using the REVOKE statement, 945
  revoking system roles using the REVOKE statement, 945
  revoking user-defined roles using the REVOKE statement, 945
ROLLBACK statement
  syntax, 950
ROLLBACK TO SAVEPOINT statement
  syntax, 951
ROLLBACK TRANSACTION statement
  Transact-SQL syntax, 951
ROLLBACK TRIGGER statement
  syntax, 952
rolling back
  transactions, 951
  transactions to savepoints, 951
  triggers, 952
ROLLUP operation
  GROUP BY clause, 847
  GROUPING function, 255
  WITH ROLLUP clause, 848
ROOT clause
  CREATE SERVER statement, 659
root_auto_unlock_time option
  ALTER LOGIN POLICY statement, 446
  CREATE LOGIN POLICY statement, 608
round earth model
  ALTER SPATIAL REFERENCE SYSTEM statement, 479
CREATE SPATIAL REFERENCE SYSTEM statement, 676
ROUND function
syntax, 351
round-off errors
about, 97
rounding errors
about, 97
DOUBLE, 100
FLOAT, 101
REAL, 104
ROW constructor
syntax, 351
Row Constructor algorithm
DUMMY system table, 1055
ROW data type
declaring, 128
ROW DELIMITED BY clause
LOAD TABLE statement, 883
UNLOAD statement, 1029
row generator
RowGenerator table (dbo), 1084
sa_rowgenerator system procedure, 1215
row limitation clause
DELETE statement, 742
SELECT statement, 957
UPDATE statement, 1039
row limits
about, 955
row locks
sa_locks system procedure, 1170
ROW LOG clause
LOAD TABLE statement, 883
ROW_NUMBER function
syntax, 353
rowcount option
setting with Transact-SQL SET statement, 987
RowGenerator
system table, 1084
ROWID function
syntax, 354
rows (see composite data types)
deleting all from a table, 1019
deleting from cursors, 739
deleting from databases, 741
fetching from cursors, 800
inserting in bulk, 873
inserting into tables, 860
inserting using cursors, 919
limiting number returned, 955
SELECT statement, 955
unloading using the UNLOAD statement, 1027
updating, 1037
ROWS clause
WINDOW clause, 1052
rs_systabgroup system role
granting using the GRANT statement, 828
revoking using the REVOKE statement, 945
RTRIM function
syntax, 356
rules
SQL language syntax, 1

S
sa_ansi_standard_packages system procedure
about, 1092
sa_audit_string system procedure
syntax, 1093
sa_certificate_info system procedure
syntax, 1094
sa_char_terms system procedure
syntax, 1095
sa_check_commit system procedure
syntax, 1096
sa_clean_database system procedure
syntax, 1097
sa_column_stats system procedure
syntax, 1100
sa_conn_activity system procedure
syntax, 1102
sa_conn_compression_info system procedure
syntax, 1104
sa_conn_info system procedure
syntax, 1106
sa_conn_list system procedure
syntax, 1109
sa_conn_options system procedure
syntax, 1110
sa_conn_properties system procedure
syntax, 1111
sa_convert_ml_progress_to_timestamp system procedure
syntax, 1113
sa_convert_timestamp_to_ml_progress system procedure
sa_copy_cursor_to_temp_table system procedure
  syntax, 1114
sa_cpu_topology system procedure
  syntax, 1115
sa_db_info system procedure
  syntax, 1118
sa_db_list system procedure
  syntax, 1119
sa_db_option system procedure
  syntax, 1120
sa_db_properties system procedure
  syntax, 1121
SA_DEBUG system role
  granting using the GRANT statement, 828
  revoking using the REVOKE statement, 945
sa_dependent_views system procedure
  syntax, 1123
sa_describe_cursor system procedure
  syntax, 1124
sa_describe_query system procedure
  syntax, 1127
sa_describe_shapefile system procedure
  syntax, 1130
sa_diagnostic_auxiliary_catalog table
  about, 1070
sa_diagnostic_blocking table
  about, 1070
sa_diagnostic_cachecontents table
  about, 1071
sa_diagnostic_connection table
  about, 1072
sa_diagnostic_cursor table
  about, 1073
sa_diagnostic_deadlock table
  about, 1074
sa_diagnostic_hostvariable table
  about, 1075
sa_diagnostic_internalvariable table
  about, 1076
sa_diagnostic_query table
  about, 1076
sa_diagnostic_request table
  about, 1078
sa_diagnostic_statement table
  about, 1079
sa_diagnostic_statistics table
  about, 1080
sa_diagnostic_tracing_level table
  about, 1081
sa_disable_auditing_type system procedure
  syntax, 1132
sa_disk_free_space system procedure
  syntax, 1133
sa_enable_auditing_type system procedure
  syntax, 1134
sa_eng_properties system procedure
  syntax, 1135
sa_error_stack_trace system procedure
  syntax, 1136
sa_event_schedules system procedure
  syntax, 1138
sa_external_libraryUnload system procedure
  syntax, 1139
sa_flush_cache system procedure
  syntax, 1140
sa_flush_statistics system procedure
  syntax, 1141
sa_get_bits system procedure
  syntax, 1141
sa_get_dtt system procedure
  syntax, 1143
sa_get_dtt_groupreads system procedure
  syntax, 1144
sa_get_histogram system procedure
  syntax, 1145
sa_get_ldapserver_status system procedure
  syntax, 1147
sa_get_request_profile system procedure
  syntax, 1148
sa_get_request_times system procedure
  syntax, 1149
sa_get_server_messages system procedure
  syntax, 1150
sa_get_table_definition system procedure
  syntax, 1151
sa_get_user_status system procedure
  syntax, 1151
sa_http_header_info system procedure
  syntax, 1154
sa_http_php_page system procedure
  syntax, 1155
sa_http_php_page_interpreted system procedure
  syntax, 1156
sa_http_variable_info system procedure
  syntax, 1158
Index

sa_index_density system procedure  
syntax, 1160

sa_index_levels system procedure  
syntax, 1162

sa_install_feature system procedure  
syntax, 1164

sa_java_loaded_classes system procedure  
syntax, 1165

sa_list_cursors system procedure  
syntax, 1166

sa_load_cost_model system procedure  
syntax, 1167

sa_locks system procedure  
syntax, 1168

sa_make_object system procedure  
syntax, 1172

sa_materialized_view_can_be_immediate system procedure  
combining results with sa_materialized_view_info, 1179  
syntax, 1174

sa_materialized_view_info system procedure  
AvailForOptimization property, 1176  
combining results with  
sa_materialized_view_can_be_immediate, 1179  
DataLastModified property, 1176  
DataStatus property, 1176  
examples, 1179  
OwnerName property, 1176  
Status property, 1176  
syntax, 1176  
ViewLastRefreshed property, 1176  
ViewName property, 1176

sa_migrate system procedure  
syntax, 1183

sa_migrate_create_fks system procedure  
syntax, 1186

sa_migrate_create_remote_fks_list system procedure  
syntax, 1187

sa_migrate_create_remote_table_list system procedure  
syntax, 1188

sa_migrate_create_tables system procedure  
syntax, 1189

sa_migrate_data system procedure  
syntax, 1191

sa_migrate_drop_proxy_tables system procedure  
syntax, 1192

sa_mirror_server_status system procedure  
syntax, 1193

sa_nchar_terms system procedure  
syntax, 1195

sa_performance_diagnostics system procedure  
syntax, 1196

sa_performance_statistics system procedure  
syntax, 1202

sa_post_login_procedure system procedure  
syntax, 1203

sa_procedure_profile system procedure  
syntax, 1204

sa_procedure_profile_summary system procedure  
syntax, 1206

sa_recompile_views system procedure  
syntax, 1209

sa_refresh_materialized_views system procedure  
syntax, 1210

sa_refresh_text_indexes system procedure  
syntax, 1211

sa_remove_tracing_data system procedure  
syntax, 1211

sa_report_deadlocks system procedure  
syntax, 1212

sa_reserved_words system procedure  
syntax, 1214

sa_reset_identity system procedure  
syntax, 1214

sa_rowgenerator system procedure  
syntax, 1215

sa_save_trace_data system procedure  
syntax, 1217

sa_send_udp system procedure  
syntax, 1218

sa_server_messages system procedure  
syntax, 1219

sa_server_option system procedure  
syntax, 1222

sa_set_http_header system procedure  
syntax, 1239

sa_set_http_option system procedure  
syntax, 1240

sa_set_soap_header system procedure  
syntax, 1244

sa_set_tracing_level system procedure  
syntax, 1245

sa_snapshots system procedure  
syntax, 1247
sa_split_list system procedure  
syntax, 1248

sa_stack_trace system procedure  
syntax, 1250

sa_statement_text system procedure  
syntax, 1252

sa_table_fragmentation system procedure  
syntax, 1253

sa_table_page_usage system procedure  
syntax, 1254

sa_table_stats system procedure  
syntax, 1255

sa_text_index_stats system procedure  
syntax, 1256

sa_text_index_vocab system procedure  
syntax, 1258

sa_text_index_vocab_nchar system procedure  
syntax, 1259

sa_transactions system procedure  
syntax, 1261

sa_unload_cost_model system procedure  
syntax, 1262

sa_user_defined_counter_add system procedure  
syntax, 1263

sa_user_defined_counter_set system procedure  
syntax, 1265

sa_validate system procedure  
syntax, 1266

sa_verify_password system procedure  
syntax, 1267

sample covariance  
about, 197

sample variance  
about, 404

SAVE OPTION VALUES clause  
ALTER TEXT CONFIGURATION statement, 502

SAVE TRANSACTION statement  
Transact-SQL syntax, 953

SAVEPOINT statement  
syntax, 954

savepoint-name  
common element in SQL syntax, 421

savepoints  
creating, 954

release, 931

rolling back to savepoints, 951

SCALE clause  
GET DESCRIPTOR statement [ESQL], 824

SET DESCRIPTOR statement [ESQL], 967

scale out

dropping servers, 764

SET MIRROR OPTION statement, 968

SCHEDULE clause  
CREATE EVENT statement, 572

scheduled events  
triggering, 1018

WAITFOR statement, 1047

scheduling  
creating events using the CREATE EVENT

statement, 570

events using the ALTER EVENT statement, 436

events using the CREATE EVENT statement, 570

WAITFOR, 1047

schema locks  
sa_locks system procedure, 1170

schemas  
creating, 654
default system views, 1347

system tables, 1055

SCRIPT VERSION clause  
CREATE SYNCHRONIZATION

SUBSCRIPTION statement [MobiLink], 687

scripted upload  
converting progress values to a TIMESTAMP,

1113

converting progress values to an UNSIGNED

BIGINT, 1114

CREATE PUBLICATION syntax, 647

SCROLL clause  
DECLARE CURSOR statement, 730

FOR statement, 805

SCROLL cursors  
declaring, 729

search conditions  
about, 40

ALL, 42

ANY, 43

BETWEEN, 45

CONTAINS, 55

EXISTS, 62

explicit selectivity estimates, 64

IN, 54

IS DISTINCT FROM, 44

IS NOT DISTINCT FROM, 44

IS NOT NULL, 62

IS NULL, 62
IS TRUE or FALSE search conditions, 63
IS UNKNOWN search condition, 63
LIKE, 47
REGEXP, 51
SIMILAR TO, 53
SOME, 43
subqueries in, 42
syntax, 40
three-valued logic, 63
truth value, 63
SEARCH DN clause
ALTER LDAP SERVER statement, 443
CREATE LDAP SERVER statement, 604
search-condition
common element in SQL syntax, 421
SECOND function
syntax, 356
SECONDS function
syntax, 357
SECURE clause
ALTER SERVICE statement [HTTP web service], 466
ALTER SERVICE statement [SOAP web service], 471
CREATE SERVICE statement [HTTP web service], 469, 666
CREATE SERVICE statement [SOAP web service], 475, 672
secure features
altering key, 1268
changing key, 1268
changing with sa_server_option, 1222
creating keys, 1273
deleting keys, 1279
SecureFeatures property
setting with sa_server_option, 1222
security
SQL Remote replication, 948, 949
SELECT
converting T-SQL, 406
select from DML
FROM clause, 815
SELECT list
describing cursors, 744
SELECT list clause
SELECT statement, 958
SELECT LIST FOR clause
DESCRIBE statement [ESQL], 744
SELECT privilege
GRANT statement, 828
REVOKE statement, 946
SELECT privilege clause
GRANT statement, 830
SELECT statement
selecting from stored procedures, 812
syntax, 955
Transact-SQL syntax, 965
selecting
for unloading using the UNLOAD statement, 1027
forming intersections, 870
forming set differences, 789
forming unions, 1025
rows, 955
selectivity estimates
source of estimates, 237
user-defined, 64
self_recursion option
setting with Transact-SQL SET statement, 987
SEND AT clause
about, 834
GRANT CONSOLIDATE statement [SQL Remote], 834
GRANT REMOTE statement [SQL Remote], 844
SEND EVERY clause
about, 834
GRANT CONSOLIDATE statement [SQL Remote], 834
GRANT REMOTE statement [SQL Remote], 844
sending
SQL statements to remote servers, 808
sending dates and times to the database
about, 109
SENSITIVE clause
DECLARE CURSOR statement, 731
FOR statement, 805
sequence generator
adding comments using the COMMENT statement, 538
ALTER SEQUENCE statement, 462
CREATE SEQUENCE statement, 655
DROP SEQUENCE statement, 771
revoking usage privileges, 946
sequences
adding comments using the COMMENT statement, 538
ALTER SEQUENCE statement, 462
CREATE SEQUENCE statement, 655
CREATE TABLE statement, 489, 695
DROP SEQUENCE statement, 771
granting usage privileges, 846
revoking usage privileges, 946
SELECT statement, 963
serializable
FROM clause, 816
SERIALIZABLE clause
REFRESH MATERIALIZED VIEW statement, 925
SERIALIZABLE table hint
FROM clause, 816
Server
HTTP_RESPONSE_HEADER function, 268
server options
setting with sa_server_option system procedure, 1222
server-option clause
ALTER MIRROR SERVER statement, 454
CREATE MIRROR SERVER statement, 618
ServerIdle system event
example, 901
servers
altering remote attributes using the ALTER SERVER statement, 463
creating, 657
creating events for idle servers using the CREATE EVENT statement, 570
dropping remote servers, 772
starting database, 996
stopping database, 1007
services
adding comments using the COMMENT statement, 538
altering HTTP services using ALTER SERVICE statement [HTTP web service], 466
altering SOAP services using ALTER SERVICE statement [SOAP web service], 471
dropping web services using the DROP SERVICE statement, 772
SessionTimeout option
sa_set_http_option system procedure, 1240
SET clause
CREATE FUNCTION statement [web clients], 590
CREATE PROCEDURE statement [web clients], 632
redirection URIs, 632
UPDATE (positioned) statement, 1033
UPDATE statement, 1039
UPDATE statement [SQL Remote], 1034
SET CONNECTION statement
embedded SQL syntax, 966
Interactive SQL syntax, 966
SET DESCRIPTOR statement
embedded SQL syntax, 967
SET HIDDEN clause
ALTER EVENT statement, 437
ALTER FUNCTION statement, 439
ALTER MATERIALIZED VIEW statement, 450
ALTER PROCEDURE statement, 456
ALTER VIEW statement, 512
SET MIRROR OPTION statement
syntax, 968
set operators
EXCEPT statement, 789
INTERSECT statement, 870
NULL, 72
set difference, 789
set intersection, 870
set union, 1025
UNION statement, 1025
SET OPTION statement
Interactive SQL syntax, 975
taxonomy, 972
Transact-SQL syntax, 987
SET PARTNER FAILOVER clause
ALTER DATABASE statement, 428
SET PERMANENT statement
Interactive SQL syntax, 975
SET REMOTE OPTION statement
common options, 979
file options, 980
ftp options, 980
http options, 981
smtp options, 982
SQL Remote syntax, 978
SET SCRIPT VERSION ... ON SUBSCRIPTION clause
START SYNCHRONIZATION SCHEMA CHANGE statement, 1003
SET SCRIPT VERSION clause
ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 484
Index

START SYNCHRONIZATION SCHEMA
CHANGE statement, 1003
SET SESSION AUTHORIZATION statement
syntax, 990
SET SQLCA statement
embedded SQL syntax, 984
SET statement
Interactive SQL syntax, 975
syntax, 985
Transact-SQL syntax, 987
SET TEMPORARY OPTION statement
Interactive SQL syntax, 975
syntax, 972
SET USER system privilege
GRANT statement, 827
granting using the GRANT statement, 829
SET_BIT function
syntax, 359
SET_BITS function
syntax, 360
sets
regular expressions, 27
setting
connections, 966
descriptor areas, 967
options, 972
options in Interactive SQL, 975
options in Transact-SQL, 987
remote options, 978
SQLCAs, 984
users, 990
values of SQL variables, 985
SETUSER statement
syntax, 990
shapefiles
loading using the INPUT statement, 854
loading using the LOAD TABLE statement, 880
loading using the st_geometry_load_shapefile system procedure, 1327
SHARE BY ALL clause
CREATE TABLE statement, 693
shutting down
databases, 1004
SIGN function
syntax, 361
SIGNAL statement
syntax, 992
signaling
errors, 920
exceptions, 936
SIMILAR function
syntax, 361
SIMILAR TO search condition
compared to REGEXP and LIKE, 45
database collation and matching, 54
matching sub-character classes, 53
regular expressions, 27
syntax, 53
SIN function
syntax, 362
SIZE clause
ATTACH TRACING statement, 514
skew
detecting in indexes using sa_index_density, 1160
SKIP clause
LOAD TABLE statement, 883
MERGE statement, 896
slash-asterisk
comment indicator, 86
slope
regression lines, 338
SMALLDATETIME data type
sending dates and times to the database, 109
syntax, 119
SMALLINT data type
comparing with dates and times, 134
syntax, 105
SMALLINT UNSIGNED data type
syntax, 105
SMALLMONEY data type
syntax, 107
SMTP
e-mail example, 1338
extended system procedures, 1086
return codes, 1086
starting email sessions, 1341
stopping email sessions, 1344
SMTP clause
GRANT CONSOLIDATE statement [SQL Remote], 834
SMTP message type
SQL Remote CREATE REMOTE MESSAGE TYPE statement, 651
snap to grid
SNAP TO GRID clause, ALTER SPATIAL REFERENCE SYSTEM statement, 479
SNAP TO GRID clause, CREATE SPATIAL REFERENCE SYSTEM statement, 678
SNAPSHOT clause
   REFRESH MATERIALIZED VIEW statement, 925
snapshot isolation
   BEGIN SNAPSHOT statement, 523
   sa_snapshots system procedure, 1247
   sa_transactions system procedure, 1261
   using with full text search, 711
snapshots
   BEGIN SNAPSHOT statement, 523
SOAP
   CREATE SERVICE statement [SOAP web service], 472, 669
SOAP functions
   alphabetical list, 151
SOAP headers
   setting, 1244
SOAP over HTTP services
   CREATE SERVICE statement [SOAP web service], 668
SOAP services
   data typing, 473, 669
SOAP system procedures
   alphabetical list, 1085
SOAP_HEADER function
   syntax, 363
SOAPHEADER clause
   CREATE FUNCTION statement [web clients], 588
   CREATE PROCEDURE statement [web clients], 634
SOME search condition
   syntax, 43
sort keys
   generating using the SORTKEY function, 364
sorting
   SORTKEY function, 364
SORTKEY function
   collation tailoring, 364
   syntax, 364
SOUNDEX function
   syntax, 367
SP
   statement indicators, 424
sp_addgroup system procedure
   Adaptive Server Enterprise system procedures, 1090
sp_addlogin system procedure
   Adaptive Server Enterprise system procedures, 1090
sp_addmessage system procedure
   about, 614
   Adaptive Server Enterprise system procedures, 1090
sp_addtype system procedure
   Adaptive Server Enterprise system procedures, 1090
sp_adduser system procedure
   Adaptive Server Enterprise system procedures, 1090
sp_alter_secure_feature_key system procedure
   syntax, 1268
sp_auth_sys_role_info system procedure
   syntax, 1269
sp_changegroup system procedure
   Adaptive Server Enterprise system procedures, 1090
sp_column_privileges catalog procedure
   Adaptive Server Enterprise catalog procedures, 1091
sp_columns catalog procedure
   Adaptive Server Enterprise catalog procedures, 1091
sp_copy_directory system procedure
   syntax, 1270
sp_copy_file system procedure
   syntax, 1271
sp_create_directory system procedure
   syntax, 1272
sp_create_secure_feature_key system procedure
   syntax, 1273
sp_delete_directory system procedure
   syntax, 1275
sp_delete_file system procedure
   syntax, 1275
sp_displayroles system procedure
   syntax, 1276
sp_drop_secure_feature_key system procedure
   syntax, 1279
sp_dropgroup system procedure
   Adaptive Server Enterprise system procedures, 1090
sp_droplogin system procedure
Adaptive Server Enterprise system procedures, 1090
sp_dropmessage system procedure
Adaptive Server Enterprise system procedures, 1090
sp_droptype system procedure
Adaptive Server Enterprise system procedures, 1090
sp_dropuser system procedure
Adaptive Server Enterprise system procedures, 1090
sp_fkeys catalog procedure
Adaptive Server Enterprise catalog procedures, 1091
sp_forward_to_remote_server system procedure syntax, 1280
sp_get_last_synchronize_result system procedure syntax, 1281
sp_getmessage system procedure
Adaptive Server Enterprise system procedures, 1090
sp_has_role system procedure syntax, 1284
sp_helptext system procedure
Adaptive Server Enterprise system procedures, 1090
sp_list_directory system procedure syntax, 1286
sp_list_secure_feature_keys system procedure syntax, 1288
sp_login_environment system procedure syntax, 1289
sp_move_directory system procedure syntax, 1289
sp_move_file system procedure syntax, 1291
sp_objectpermission system procedure syntax, 1292
sp_parse_json system procedure syntax, 1295
sp_password system procedure
Adaptive Server Enterprise system procedures, 1090
sp_pkeys catalog procedure
Adaptive Server Enterprise catalog procedures, 1091
sp_proc_priv system procedure syntax, 1297
sp_remote_columns system procedure syntax, 1298
sp_remote_exported_keys system procedure syntax, 1300
sp_remote_imported_keys system procedure syntax, 1302
sp_remote_pcols system procedure syntax, 1303
sp_remote_primary_keys system procedure syntax, 1305
sp.remote_procedures system procedure syntax, 1307
sp_remote_tables system procedure syntax, 1308
sp_servercaps system procedure syntax, 1309
sp_special_columns catalog procedure
Adaptive Server Enterprise catalog procedures, 1091
sp_sproc_columns catalog procedure
Adaptive Server Enterprise catalog procedures, 1091
sp_statistics catalog procedure
Adaptive Server Enterprise catalog procedures, 1091
sp_stored_procedures catalog procedure
Adaptive Server Enterprise catalog procedures, 1091
sp_sys_priv_role_info system procedure syntax, 1311
sp_tables catalog procedure
Adaptive Server Enterprise catalog procedures, 1091
sp_trace_event_fields system procedure syntax, 1312
sp_trace_event_session_events system procedure syntax, 1314
sp_trace_event_session_target_options system procedure syntax, 1315
sp_trace_event_sessions system procedure syntax, 1316
sp_trace_events system procedure syntax, 1317
sp_tsql_environment system procedure syntax, 1318
sp_tsql_environment system procedure syntax, 1320
sp_use_secure_feature_key system procedure
  syntax, 1321
SPACE function
  syntax, 368
spatial data
  ALTER SPATIAL REFERENCE SYSTEM
    statement, 477
  CREATE SPATIAL REFERENCE SYSTEM
    statement, 673
  CREATE SPATIAL UNIT OF MEASURE
    statement, 681
  disassembling geometries, 1322
  DROP SPATIAL REFERENCE SYSTEM
    statement, 773
  DROP SPATIAL UNIT OF MEASURE statement, 774
  importing ESRI shapefiles, 880
  ISYSSPATIALREFERENCESYSTEM system table, 1066
  ISYSUNITOFMEASURE system table, 1068
  st_geometry_load_shapefile system procedure, 1327
  TREAT function, 391
  troubleshooting invalid geometries, 1322
spatial reference systems
  adding comments using the COMMENT statement, 538
  installing predefined, 1164
special character classes
  regular expressions, 30
special characters
  allowed syntax in a full text query string, 60
  SQL strings, 8
  used in binary, 6
  used in strings, 8
special tables
  about, 1055
special values
  about, 65
  CREATE TABLE statement, 489, 694
  CURRENT DATABASE, 65
  CURRENT DATE, 66
  CURRENT PUBLISHER, 66
  CURRENT REMOTE USER, 67
  CURRENT TIME, 67
  CURRENT TIMESTAMP, 68
  CURRENT USER, 69
  CURRENT UTC TIMESTAMP, 70
  CURRENT_TIMESTAMP, 68
  CURRENT_USER, 69
  LAST USER, 71
  NULL, 71
  SQLCODE, 74
  SQLSTATE, 74
  TIMESTAMP, 76
  USER, 77
  UTC_TIMESTAMP, 78
special views
  about, 1347
special-value
  common element in SQL syntax, 422
Specification property
  DB_EXTENDED_PROPERTY function, 212
SQL
  alphabetical list of SQL Anywhere Server
    statements, 424
SQL Anywhere Native
  ODBC driver reference, 659
SQL descriptor area
  INCLUDE statement, 853
  inserting rows using cursors, 919
SQL descriptor areas
  DESCRIBE statement [ESQL], 744
SQL Flagger
  SQLFLAGGER function, 369
  testing a SQL statement for non-core extensions, 1092
SQL functions
  aggregate, 143
  bit array, 145
  data type conversion, 146
  date and time, 146
  HTTP, 151
  image, 155
  introduction, 143
  miscellaneous, 150
  numeric, 151
  ranking, 145
  return NULL if you specify NULL argument, 143
  SOAP, 151
  string, 152
  system, 154
  text, 155
  types of functions, 143
  user defined, 148
SQL keywords
logical operators, 10
named parameters, 87
NULL value, 71
operator precedence, 20
operators, 9
predicates, 40
REGEXP search condition, 51
search conditions, 40
SIMILAR TO search condition, 53
SOME search condition, 43
special values, 65
SQLCODE special value, 74
SQLSTATE special value, 74
string operators, 11
strings, 6
subqueries, 23
subqueries in search conditions, 42
three-valued logic, 63
TIMESTAMP special value, 76
Transact-SQL expression compatibility, 39
USER special value, 77
UTC TIMESTAMP special value, 78
variables, 79
SQL to Java data type conversion
about, 141
SQL variables
creating, 722
declaring, 736
dropping using the DROP VARIABLE statement, 787
setting values, 985
SQL/1999
testing SQL compliance, 369
SQL/1999Q2
testing SQL compliance, 369
SQL/2003
testing SQL compliance, 369
SQL/2008
testing SQL compliance, 369
SQLCA
INCLUDE statement, 853
SQLCA clause
INCLUDE statement, 853
SQLCAs
setting, 984
SQLCODE
special value, 74
SQLDA
allocating memory for, 424
deallocating, 727
DESCRIBE statement [ESQL], 744
EXECUTE statement, 794
getting information from, 824
inserting rows using cursors, 919
setting, 967
UPDATE (positioned) statement, 1032
SQLDA clause
INCLUDE statement, 853
SQLDescribeCol
describing CHAR data types, 89
describing NCHAR data types, 92
describing NVARCHAR data types, 93
describing VARCHAR data types, 95
SQLDIALECT function
syntax, 368
SQLERROR clause
WHENEVER statement [ESQL], 1049
SQLFLAGGER function
syntax, 369
SQLSetConnectAttr
using with MESSAGE TO CLIENT, 902
SQLSTATE
conformance with ISO/ANSI standard, 74
special value, 74
SQLWARNING clause
WHENEVER statement [ESQL], 1049
SQRT function
syntax, 370
square brackets
database objects, 4
SQL identifiers, 4
square root function
SQRT function, 370
SRIDs
choosing a SRID when creating a spatial reference system, 674
ST_GEOMETRY_COLUMNS
consolidated view, 1419
st_geometry_dump system procedure
syntax, 1322
st_geometry_load_shapefile system procedure
syntax, 1327
ST_SPATIAL_REFERENCE_SYSTEMS
consolidated view, 1420
ST_UNITS_OF_MEASURE
consolidated view, 1423
STACK_TRACE function
  syntax, 371
standard deviation
  STDDEV function, 372
  STDDEV_POP function, 372
  STDDEV_SAMP function, 373
START AT clause
  DELETE statement, 742
  SELECT statement, 957
  UPDATE statement, 1039
START DATABASE statement
  syntax, 993
START DATE clause
  ALTER EVENT statement, 437
  CREATE EVENT statement, 573
START EXTERNAL ENVIRONMENT statement
  syntax, 997
START JAVA statement
  syntax, 998
START LOGGING statement
  Interactive SQL syntax, 999
START SERVER statement
  Interactive SQL syntax, 996
START SUBSCRIPTION statement
  SQL Remote syntax, 1000
START SYNCHRONIZATION DELETE statement
  MobiLink syntax, 1001
START SYNCHRONIZATION SCHEMA CHANGE statement
  MobiLink syntax, 1002
START TIME clause
  ALTER EVENT statement, 437
  CREATE EVENT statement, 572
START WITH clause
  CREATE SEQUENCE statement, 656
starting
  creating events using the CREATE EVENT statement, 570
database servers, 996
  databases, 993
e external environments using START EXTERNAL ENVIRONMENT statement, 997
Java VM using the START JAVA statement, 998
logging in Interactive SQL, 999
  passthrough mode, 913
SQL Remote subscriptions during database extraction, 932
  subscriptions, 1000
state information files
  ALTER MIRROR SERVER statement, 454
  CREATE MIRROR SERVER statement, 618
state_file option
  ALTER MIRROR SERVER statement, 454
  CREATE MIRROR SERVER statement, 618
statement label
  common element in SQL syntax, 422
statement labels
  GOTO Transact-SQL statement, 827
statement syntax
  alphabetical list of SQL Anywhere Server statements, 424
documentation conventions, 421
statement-list
  common element in SQL syntax, 422
statements
  alphabetical list of SQL Anywhere Server statements, 424
dropping prepared statements, 775
executing prepared, 794
grouping in the BEGIN statement, 523
grouping in the BEGIN statement [TSQL], 526
grouping in the TRY statement, 1022
preparing, 915
static cursors
  declaring, 729
statistics
  CREATE STATISTICS statement, 682
dropping using the DROP STATISTICS statement, 775
  flushing to disk, 1141
  loading, 873
  only partially updated by LOAD TABLE, 886
retrieving using sa_get_histogram system procedure, 1145
SYSCOLSTAT system view, 1351
updating using the ALTER SERVICE statement, 480
STATISTICS clause
  LOAD TABLE statement, 885
statistics cleaner
  controlling with sa_server_option, 1222
StatisticsCleaner property
  setting with sa_server_option, 1222
Status property
  sa_materialized_view_info system procedure, 1176
STDDEV function
ALTER SPATIAL REFERENCE SYSTEM statement, 479
CREATE SPATIAL REFERENCE SYSTEM statement, 679
stored functions
  native function calls, 582
stored procedures
  converting T-SQL, 406
  creating, 639
  creating in Transact-SQL, 637
  creating web services, 628
  executing in dynamic SQL, 791
  executing in Transact-SQL, 796
  INPUT statement cannot be used, 859
native function calls, 624
selecting from, 812
system procedures, 1085
STR function
  syntax, 375
STRING function
  syntax, 376
string functions
  alphabetical list, 152
string length
  LENGTH function, 283
string literals
  about, 8
  escape sequences, 8
  special characters, 8
string operators
  syntax, 11
string position
  LOCATION function, 287
string-expression
  common element in SQL syntax, 422
string_truncation option
  setting with Transact-SQL SET statement, 987
strings
  about, 6
  ambiguous conversions of two-digit years, 111
  changing the interpretation of delimited strings, 39
  converting to dates, 136
  delimiter, 39
  escape characters, 8
  quotation marks, 39
  removing trailing blanks, 356
  replacing, 345
  SQL functions, 152
Transact-SQL, 39
STRIP clause
  LOAD TABLE statement, 883
strong encryption
  CREATE DATABASE statement, 553
STRTOUUID function
  syntax, 377
STUFF function
  syntax, 378
su
  setting users, 990
sub-character classes
  REGEXP search condition, 51
  regular expressions, 30
  SIMILAR TO search condition, 53
SUBDIRS clause
  CREATE SERVER statement, 659
subqueries
  evaluate to NULL if no matching rows, 23
  in SQL search conditions, 42
  syntax, 23
SUBSCRIBE BY clause
  ALTER PUBLICATION statement, 457
  CREATE PUBLICATION statement [MobiLink][SQL Remote], 648
  UPDATE statement, 1037
  UPDATE statement [SQL Remote], 1034
subscriptions
  ALTER SYNCHRONIZATION SUBSCRIPTION statement, 482
  CREATE SUBSCRIPTION statement [SQL Remote], 683
  CREATE SYNCHRONIZATION SUBSCRIPTION statement, 686
  DROP SUBSCRIPTION statement, 776
  DROP SYNCHRONIZATION SUBSCRIPTION statement, 778
  SQL Remote creation, 683
  SQL Remote REMOTE RESET statement, 932
  SQL Remote UPDATE statement, 1036
  START SUBSCRIPTION statement [SQL Remote], 1000
  STOP SUBSCRIPTION statement [SQL Remote], 1008
  SYNCHRONIZE SUBSCRIPTION statement [SQL Remote], 1016
  UPDATE statement, 1040
substitution characters
  comparisons between CHAR and NCHAR, 132
  lossy conversions, 132
SUBSTR function
  syntax, 379
SUBSTRING function
  syntax, 379
  substrings
    about, 379
    replacing, 345
SUM function
  syntax, 381
  super types
    about, 131
SUSER_ID function
  syntax, 382
SUSER_NAME function
  syntax, 383
SWITCHOFFSET function
  syntax, 383
synchronization_mode option
  SET MIRROR OPTION statement, 969
SYNCHRONIZE statement
  MobiLink syntax, 1011
  SQL Remote syntax, 1016
synchronizing subscriptions
  SYNCHRONIZE SUBSCRIPTION statement [SQL Remote], 1016
  syntax
    arithmetic operators, 11
    array operators, 18
    bitwise operators, 19
    CASE expression, 24
    column names, 23
    comments, 85
    comparison operators, 9
    connection-level variables, 81
    constants, 6
    constants in expressions, 22
    conventions, 422
    CURRENT DATABASE special value, 65
    CURRENT DATE special value, 66
    CURRENT PUBLISHER special value, 66
    CURRENT TIMESTAMP special value, 68
    CURRENT USER special value, 69
    CURRENTUTC TIMESTAMP special value, 70
    CURRENT_TIMESTAMP special value, 68
    CURRENT_USER special value, 69
documentation conventions, 421
IF expressions, 23
IS NULL search condition, 62
IS TRUE or FALSE search condition, 63
LAST USER special value, 71
LEN function, 283
list of SQL reserved words, 1
local variables, 80
logical operators, 10
named parameters, 87
NULL value, 71
predicates, 40
search conditions, 40
special values, 65
SQL CURRENT REMOTE USER special value, 67
SQL CURRENT TIME special value, 67
SQL expressions, 21
SQL functions, 143
SQL identifiers, 4
SQL keywords, 1
SQL operator precedence, 20
SQL operators, 9
SQL statements, 424
SQL subqueries, 23
SQL subqueries in search conditions, 42
SQL variables, 79
SQLCODE special value, 74
SQLSTATE special value, 74
string operators, 11
strings, 6
testing compliance against a standard, 369
three-valued logic, 63
TIMESTAMP special value, 76
Transact-SQL expression compatibility, 39
USER special value, 77
UTC TIMESTAMP special value, 78
syntax conventions
SQL statements, 422
SYS
default system views, 1347
system tables, 1055
SYS system role
granting using the GRANT statement, 828
revoking using the REVOKE statement, 945
SYS_AUTH_BACKUP_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
sys_auth_validate_role compatibility role
granting compatibility roles using the GRANT statement, 828
SYS_AUTH_DBA_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_PROFILE_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_READCLIENTFILE_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_READFILE_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_RESOURCE_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_SSO_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_SA_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_SSO_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_SSO_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_VALIDATE_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
sys_auth_validate_role compatibility role
granting compatibility roles using the GRANT statement, 828
SYS_AUTH_WRITECLIENTFILE_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_AUTH_WRITEFILE_ROLE compatibility role
granting compatibility roles using the GRANT statement, 828
revoking compatibility roles using the REVOKE statement, 945
SYS_REPLICATION_ADMIN_ROLE system role
granting, 840
granting using the GRANT statement, 828
revoking using the REVOKE statement, 945
SYS_RUN_REPLICATION_ROLE system role
granting, 842
granting using the GRANT statement, 828
revoking using the REVOKE statement, 945
SYS_SAMONITOR_ROLE system role
granting using the GRANT statement, 828
revoking using the REVOKE statement, 945
SYS_SPATIAL_ADMIN_ROLE system role
granting using the GRANT statement, 828
revoking using the REVOKE statement, 945
SYSARTICLE
system view, 1348
SYSARTICLECOL
system view, 1348
SYSARTICLECOLS
consolidated view, 1423
SYSARTICLES
consolidated view, 1424
SYSCAPABILITIES
consolidated view, 1424
SYSCAPABILITY
system view, 1349
SYSCAPABILITYNAME
system view, 1349
SYSCATALOG
consolidated view, 1425
SYSCERTIFICATE
adding certificates, 547
system view, 1350
SYSCHECK
system view, 1350
SYSCOLAUTH
consolidated view, 1425
SYSCOLLATION
compatibility view (deprecated), 1439
SYSCOLLATIONMAPPINGS
compatibility view (deprecated), 1439
SYSCOLPERM
system view, 1351
SYSCOLSTAT
system view, 1351
SYSCOLSTATS
consolidated view, 1426
SYSCOLUMN
compatibility view (deprecated), 1439
SYSCOLUMNS
consolidated view, 1426
syscolumns
  dbo-owned view, 1449
syscomments
  dbo-owned view, 1449
SYSCONSTRAINT
system view, 1352
SYSDATETIMEOFFSET function
  syntax, 384
SYSDBFILE
system view, 1353
SYSDBSPACE
system view, 1354
SYSDBSPACEPERM
system view, 1354
SYSDependency
system view, 1355
SYSDomain
system view, 1356
SYSEVENT
system view, 1356
SYSEVENTTYPE
system view, 1358
SYSEXTERNENV
system view, 1358
SYSEXTERNENVOBJECT
system view, 1360
SYSEXTERNLOGIN
system view, 1361
SYSFDATE
compatibility view (deprecated), 1440
SYSFKEY
compatibility view (deprecated), 1441
SYSFKEY
system view, 1361
SYSEXTERNALKEY
compatibility view (deprecated), 1441
SYSEXTERNALKEYS
consolidated view, 1427
SYSGROUP  
compatibility view, 1442
SYSGROUPS  
compatibility view, 1442
SYSHISTORY  
system view, 1362
SYSIDX  
system view, 1364
SYSIDXCOL  
system view, 1366
SYSINDEX  
compatibility view (deprecated), 1443
SYSINDEXES  
consolidated view, 1428
sysindexes  
dbo-owned view, 1449
SYSINFO  
compatibility view (deprecated), 1443
SYSIXCOL  
compatibility view (deprecated), 1444
SYSJAR  
system view, 1367
SYSJARCOMPONENT  
��统视图, 1367
SYSJAVAACLASS  
system view, 1368
SYSLDAPSERVER  
system view, 1369
SYSLOGINMAP  
system view, 1370
SYSLOGINPOLICY  
system view, 1371
SYSLOGINPOLICYOPTION  
system view, 1371
syslogins  
dbo-owned view, 1449
SYSMIRROROPTION  
system view, 1372
SYSMIRRORSERVER  
system view, 1372
SYSMIRRORSERVEROPTION  
system view, 1373
SYSMVOPTION  
system view, 1373
SYSMVOPTIONNAME  
system view, 1374
SYSOBJECT  
system view, 1375
sysobjects  
dbo-owned view, 1449
SYSOPTION  
system view, 1375
SYSOPTIONS  
consolidated view, 1428
SYSOPTSTAT  
system view, 1376
SYSPHYSIDX  
system view, 1376
SYSPROCAUTH  
consolidated view, 1429
SYSPROCEDURE  
系统视图, 1377
SYSPROCSPARM  
系统视图, 1377
SYSPROCSPARMS  
consolidated view, 1429
SYSPROCSPERM  
系统视图, 1380
SYSPROCS  
consolidated view, 1430
SYSPROXYTAB  
系统视图, 1380
SYSPUBLICATION  
系统视图, 1381
SYSPUBLICATIONS  
consolidated view, 1430
SYSREMARK  
系统视图, 1382
SYSREMOTEOPTION  
系统视图, 1383
SYSREMOTEOPTION2  
consolidated view, 1431
SYSREMOTEOPTIONS  
consolidated view, 1431
SYSREMOTEOPTIONTYPE  
系统视图, 1383
SYSREMOTETYPE  
系统视图, 1383
SYSREMOTETYPES  
consolidated view, 1432
SYSREMOTETYPES  
系统视图, 1384
SYSREMOTEUSERS  
consolidated view, 1432
SYSROLEGRANT  
系统视图, 1386
Index

SYSROLEGRANTTEXT
  system view, 1387
SYSROLEGRANTS
  system view, 1387
SYSSCHEDULE
  system view, 1388
SYSSEQUENCE
  system view, 1389
SYSSEQUENCEPERM
  system view, 1390
SYSSERVER
  system view, 1391
SYSSOURCE
  system view, 1391
SYSSPATIALREFERENCESYSTEM
  system view, 1392
SYSSQLSERVERVERTYPE
  system view, 1395
SYSSUBSCRIPTION
  system view, 1395
SYSSUBSCRIPTIONS
  consolidated view, 1433
SYSSYNC
  system view, 1396
SYSSYNC2
  consolidated view, 1433
SYSSYNCPROFILE
  system view, 1397
SYSSYNCPUBLICATIONDEFAULTS
  consolidated view, 1434
SYSSYNCS
  consolidated view, 1434
SYSSYNCSCRIPT
  system view, 1398
SYSSYNCSCRIPTS
  consolidated view, 1435
SYSSYNCSUBSCRIPTIONS
  consolidated view, 1435
SYSSYNCSUSERS
  consolidated view, 1436
SYSTAB
  compatibility view (deprecated), 1444
  compatibility view information, 1055
  system view, 1399
SYSTABAUTH
  consolidated view, 1436
SYSTABCOL
  system view, 1401
SYSTABLE
  system view, 1404
system and catalog stored procedures
  about, 1092
  system calls
    from stored procedures, 1328
    xp_cmdshell system procedure, 1328
  system catalog
    tables, 1055
    views, 1347
  system extended procedures
    about, 1086
  system functions
    alphabetical list, 154
    compatibility, 154
  system privileges
    granting using the GRANT statement, 828
    revoking using the REVOKE statement, 946
  SYSTEM PROCEDURE AS clause
    ALTER DATABASE statement, 426
    CREATE DATABASE statement, 639
  SYSTEM PROCEDURE AS DEFINER clause
    ALTER DATABASE statement, 427
    CREATE DATABASE statement, 554
system procedures
  about, 1085
  Adaptive Server Enterprise system procedures, 1090
  alphabetical list, 1092
  creating messages, 614
  extended list, 1086
  HTTP, 1085
  named parameters allowed when calling, 530
  SOAP, 1085
  Sybase Central, 1085
  Transact-SQL, 1090
  Transact-SQL list, 1089
  viewing definitions, 1085
system roles
  granting using the GRANT statement, 828
  only MANAGE ROLES can administer, 829
  revoking using the REVOKE statement, 945
SYSTEM statement
  Interactive SQL syntax, 1017
  not supported by dbisqlc, 1017
system tables
  about, 1055

1538Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
<table>
<thead>
<tr>
<th>System View名称</th>
<th>页码</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUMMY</td>
<td>1055</td>
</tr>
<tr>
<td>Java</td>
<td>1084</td>
</tr>
<tr>
<td>RowGenerator</td>
<td>1084</td>
</tr>
<tr>
<td>updates</td>
<td>1055</td>
</tr>
<tr>
<td>viewing system table data</td>
<td>1055</td>
</tr>
<tr>
<td>views</td>
<td>1055</td>
</tr>
<tr>
<td><strong>系统视图</strong></td>
<td></td>
</tr>
<tr>
<td>about</td>
<td>1347</td>
</tr>
<tr>
<td>alphabetical list of system views</td>
<td>1347</td>
</tr>
<tr>
<td>SYSARTICLE</td>
<td>1348</td>
</tr>
<tr>
<td>SYSARTICLECOL</td>
<td>1348</td>
</tr>
<tr>
<td>SYSCAPABILITY</td>
<td>1349</td>
</tr>
<tr>
<td>SYSCAPABILITYNAME</td>
<td>1349</td>
</tr>
<tr>
<td>SYSCERTIFICATE</td>
<td>1350</td>
</tr>
<tr>
<td>SYSCHECK</td>
<td>1350</td>
</tr>
<tr>
<td>SYSCOLPERM</td>
<td>1351</td>
</tr>
<tr>
<td>SYSCOLSTAT</td>
<td>1351</td>
</tr>
<tr>
<td>SYSCONSTRAINT</td>
<td>1352</td>
</tr>
<tr>
<td>SYSDBFILE</td>
<td>1353</td>
</tr>
<tr>
<td>SYSDBSPACE</td>
<td>1354</td>
</tr>
<tr>
<td>SYSDBSPACEPERM</td>
<td>1354</td>
</tr>
<tr>
<td>SYSDEPENDENCY</td>
<td>1355</td>
</tr>
<tr>
<td>SYSDOMAIN</td>
<td>1356</td>
</tr>
<tr>
<td>SYSEVENT</td>
<td>1356</td>
</tr>
<tr>
<td>SYSEVENTTYPE</td>
<td>1358</td>
</tr>
<tr>
<td>SYSEXTERNENV</td>
<td>1358</td>
</tr>
<tr>
<td>SYSEXTERNENVOBJECT</td>
<td>1360</td>
</tr>
<tr>
<td>SYSEXTERNLOGIN</td>
<td>1361</td>
</tr>
<tr>
<td>SYSFILE</td>
<td>1440</td>
</tr>
<tr>
<td>SYSFKEY</td>
<td>1361</td>
</tr>
<tr>
<td>SYSHISTORY</td>
<td>1362</td>
</tr>
<tr>
<td>SYSIDX</td>
<td>1364</td>
</tr>
<tr>
<td>SYSIDXCOL</td>
<td>1366</td>
</tr>
<tr>
<td>SYSJAR</td>
<td>1367</td>
</tr>
<tr>
<td>SYSJARCOMPONENT</td>
<td>1367</td>
</tr>
<tr>
<td>SYSJAVACLASS</td>
<td>1368</td>
</tr>
<tr>
<td>SYSLDAPSERVER</td>
<td>1369</td>
</tr>
<tr>
<td>SYSLOGINMAP</td>
<td>1370</td>
</tr>
<tr>
<td>SYSLOGINPOLICY</td>
<td>1371</td>
</tr>
<tr>
<td>SYSLOGINPOLICYOPTION</td>
<td>1371</td>
</tr>
<tr>
<td>SYSMIRROROPTION</td>
<td>1372</td>
</tr>
<tr>
<td>SYSMIRRORSERVER</td>
<td>1372</td>
</tr>
<tr>
<td>SYSMIRRORSERVEROPTTION</td>
<td>1373</td>
</tr>
<tr>
<td>SYSMVOPTION</td>
<td>1373</td>
</tr>
<tr>
<td>SYSMVOPTIONNAME</td>
<td>1374</td>
</tr>
<tr>
<td>SYSOBJECT</td>
<td>1375</td>
</tr>
<tr>
<td>SYSOPTION</td>
<td>1375</td>
</tr>
<tr>
<td>SYSOPTSTAT</td>
<td>1376</td>
</tr>
<tr>
<td>SYSPHYSIDX</td>
<td>1376</td>
</tr>
<tr>
<td>SYSPROCEDURE</td>
<td>1377</td>
</tr>
<tr>
<td>SYSPROCPARM</td>
<td>1378</td>
</tr>
<tr>
<td>SYSPROCPERM</td>
<td>1380</td>
</tr>
<tr>
<td>SYSPROXYTAB</td>
<td>1380</td>
</tr>
<tr>
<td>SYSPUBLICATION</td>
<td>1381</td>
</tr>
<tr>
<td>SYREMARK</td>
<td>1382</td>
</tr>
<tr>
<td>SYSREMOOTEPTION</td>
<td>1383</td>
</tr>
<tr>
<td>SYSREMOOTEPTIONTYPE</td>
<td>1383</td>
</tr>
<tr>
<td>SYSSREMOTEUSER</td>
<td>1384</td>
</tr>
<tr>
<td>SYSSROLEGANT</td>
<td>1386</td>
</tr>
<tr>
<td>SYSSROLEGANTTEXT</td>
<td>1387</td>
</tr>
<tr>
<td>SYSSROLEGANTS</td>
<td>1387</td>
</tr>
<tr>
<td>SYSSCHEDULE</td>
<td>1388</td>
</tr>
<tr>
<td>SYSEQUENCE</td>
<td>1389</td>
</tr>
<tr>
<td>SYSEQUENCEPERM</td>
<td>1390</td>
</tr>
<tr>
<td>SYSSERVER</td>
<td>1391</td>
</tr>
<tr>
<td>SYSSOURCE</td>
<td>1391</td>
</tr>
<tr>
<td>SYSSPATIALREFERENCESYSTEM</td>
<td>1392</td>
</tr>
<tr>
<td>SYSQLSERVERTYPE</td>
<td>1395</td>
</tr>
<tr>
<td>SYSSUBSCRIPTION</td>
<td>1395</td>
</tr>
<tr>
<td>SYSSYNC</td>
<td>1396</td>
</tr>
<tr>
<td>SYSSYNCPROFILE</td>
<td>1397</td>
</tr>
<tr>
<td>SYSSYNCSCRIPT</td>
<td>1398</td>
</tr>
<tr>
<td>SYSTAB</td>
<td>1399</td>
</tr>
<tr>
<td>SYSTABCOL</td>
<td>1401</td>
</tr>
<tr>
<td>SYSTABLEPERM</td>
<td>1404</td>
</tr>
<tr>
<td>SYSTEXTIDX</td>
<td>1407</td>
</tr>
<tr>
<td>SYSTEXTIDXTAB</td>
<td>1408</td>
</tr>
<tr>
<td>SYSTRIGGER</td>
<td>1409</td>
</tr>
<tr>
<td>SYSTYPEDMAP</td>
<td>1411</td>
</tr>
<tr>
<td>SYSUNITOFMEASURE</td>
<td>1411</td>
</tr>
<tr>
<td>SYSUSER</td>
<td>1412</td>
</tr>
<tr>
<td>SYSUSERMESSAGE</td>
<td>1414</td>
</tr>
<tr>
<td>SYUSERTYPE</td>
<td>1415</td>
</tr>
<tr>
<td>SYSVIEW</td>
<td>1416</td>
</tr>
<tr>
<td>SYSWEBSERVICE</td>
<td>1418</td>
</tr>
<tr>
<td>SYSTEXTCONFIG</td>
<td>1405</td>
</tr>
<tr>
<td>SYSTEXTIDXTAB</td>
<td>1408</td>
</tr>
<tr>
<td>SYSTRIGGER</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.01539
system view, 1409
SYSTRIGGERS
    consolidated view, 1437
SYSTYPEMAP
    system view, 1411
systypes
    dbo-owned view, 1449
SYSUNITOFMEASURE
    system view, 1411
SYSUSER
    system view, 1412
SYSUSERAUTH
    compatibility view (deprecated), 1446
SYSUSERAUTHORITY
    compatibility view (deprecated), 1446
SYSUSERLIST
    compatibility view (deprecated), 1447
SYSUSERMESSAGE
    system view, 1414
SYSUSEROPTIONS
    consolidated view, 1438
SYSUSERPERM
    compatibility view (deprecated), 1447
SYSUSERPERMS
    compatibility view (deprecated), 1448
sysusers
    dbo-owned view, 1449
SYSUSERTYPE
    system view, 1415
SYSVIEW
    system view, 1416
SYSVIEWS
    consolidated view, 1438
    consolidated view information, 1055
SYSWEBSERVICE
    system view, 1418

T
TABLE clause
    ALTER PUBLICATION statement, 457
    CREATE PUBLICATION statement [MobiLink] [SQL Remote], 647
    DESCRIBE statement, 748
    TRUNCATE statement, 1019
    VALIDATE statement, 1045
table columns
    listing in Interactive SQL, 747
table constraints
    adding using the ALTER TABLE statement, 493
    adding, deleting, or altering using the ALTER TABLE statement, 486
    changing using ALTER TABLE statement, 496
    CREATE TABLE statement, 697
table decryption
    ALTER TABLE statement, 486
table encryption
    ALTER TABLE statement, 486
    CREATE ENCRYPTED TABLE DATABASE statement, 564
table hints
    FROM clause, 816
table indexes
    listing in Interactive SQL, 747
table list
    FROM clause, 812
table locks
    sa_locks system procedure, 1170
table number
    system views, 1399
table pages
    setting PCTFREE, 690
    setting PCTFREE using the ALTER TABLE statement, 486
    setting PCTFREE using the CREATE LOCAL TEMPORARY TABLE statement, 606
    setting PCTFREE using the LOAD TABLE statement, 873
table-list
    common element in SQL syntax, 422
table-name
    common element in SQL syntax, 422
TABLE.COLUMN clause
    DESCRIBE statement [ESQL], 744
tables
    adding comments using the COMMENT statement, 538
    ALTER TABLE statement, 486
    altering using the ALTER TABLE statement, 486
    bulk loading, 873
    CREATE TABLE statement, 690
    creating local temporary, 735
    creating local temporary tables using the CREATE LOCAL TEMPORARY TABLE statement, 606
    creating proxy tables using the CREATE EXISTING TABLE statement, 576
determining dependencies, 1123
dropping using the DROP TABLE statement, 780
encrypting with CREATE ENCRYPTED TABLE DATABASE statement, 564
exporting data into files from, 907
generating definition with sa_get_table_definition system procedure, 1151
importing data from files into, 854
inserting rows into, 860
locking, 890
renaming, 496
reorganizing, 935
SQL Remote updating, 1034
truncating, 1019
unloading using the UNLOAD statement, 1027

TAN function
syntax, 385

tape drives
creating database backups using the BACKUP statement, 516

TempFreePercent event condition
about, 238

TempFreeSpace event condition
about, 238

TEMPORARY clause
ALTER DBSPACE statement, 433
CREATE PROCEDURE statement, 639
temporary dbspace
calibrating, 427
temporary files
determining available space, 1133
temporary functions
about, 595

TEMPORARY keyword
CREATE FUNCTION statement [user defined], 595
temporary options
SET OPTION statement, 972
setting in Interactive SQL, 975
temporary procedures
CREATE PROCEDURE statement, 639
temporary stored procedures
creating, 639
temporary tables
CREATE TABLE statement, 690
CREATE TABLE usage, 701
creating local temporary files using the CREATE LOCAL TEMPORARY TABLE statement, 606
declaring local, 735
Transact-SQL CREATE TABLE statement, 702
views disallowed on local, 725

TempSize event condition
about, 238

TERM BREAKER clause
ALTER TEXT CONFIGURATION statement, 502
TERM BREAKER EXTERNAL NAME clause
ALTER TEXT CONFIGURATION statement, 501
term breakers
specifying an external term breaker library, 502
using sa_char_terms to test how strings are broken into terms, 1095
using sa_nchar_terms to test how strings are broken into terms, 1195
warning about using non-alphanumerics in query string, 58
terms
MAXIMUM TERM LENGTH clause, 501
MINIMUM TERM LENGTH clause, 501
TERM BREAKER clause, 502
text
reading from the database using the READTEXT statement, 924
text configuration objects
adding comments using the COMMENT statement, 538
ALTER TEXT CONFIGURATION statement, 501
CREATE TEXT CONFIGURATION statement, 709
dropping, 781
using sa_char_terms to test how strings are broken into terms, 1095
using sa_nchar_terms to test how strings are broken into terms, 1195
TEXT data type
syntax, 94
text functions
about, 155
text indexes
adding comments using the COMMENT statement, 538
ALTER TEXT INDEX statement, 504
CREATE TEXT INDEX statement, 711
creation options stored in SYSMVOPTION, 1373
dropping, 782
listing using sa_text_index_stats, 1256
refreshing using REFRESH TEXT INDEX, 928
refreshing using sa_refresh_text_indexes, 1211
truncating, 1021
validating, 1045
TEXTPTTR function
   syntax, 385
textsize option
   setting with Transact-SQL SET statement, 987
THEN
   IF expressions, 23
THEN clause
   IF statement, 850
three-valued logic
   NULL value, 72
   syntax, 63
TIME clause
   WAITFOR statement, 1048
TIME data type
   comparing with dates and times, 134
   sending dates and times to the database, 109
   syntax, 120
time data types
   DATETIME, 116
   DATETIMEOFFSET, 117
   overview, 108
   SMALLDATETIME, 119
   TIMESTAMP, 121
   TIMESTAMP WITH TIME ZONE, 123
time functions
   alphabetical list, 146
time of day
   storing, 109
time zone
   CURRENT UTC TIMESTAMP, 70
   UTC TIMESTAMP, 78
times
   comparing, 134
   conversion functions, 146
   queries, 114
   sending to the database, 109
TIMESTAMP clause
   CREATE TABLE statement, 690
TIMESTAMP data type
   comparing with dates and times, 134
   sending dates and times to the database, 109
   syntax, 121
TIMESTAMP default
   CREATE TABLE statement, 489, 694
TIMESTAMP special value
   syntax, 76
TIMESTAMP WITH TIME ZONE data type
   comparing with dates and times, 134
   sending dates and times to the database, 109
   syntax, 123
timestamps
   comparing, 134
   comparisons using TIMESTAMP WITH TIMEZONE, 123
   conversions using TIMESTAMP WITH TIMEZONE, 124
TINYINT data type
   syntax, 105
TINYINT UNSIGNED data type
   syntax, 105
TLS clause
   ALTER LDAP SERVER statement, 444
   CREATE LDAP SERVER statement, 605
TO clause
   ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 483
   CONNECT statement, 543
   CREATE EXTERNLOGIN statement, 578
   CREATE SUBSCRIPTION statement [SQL Remote], 683
   CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink], 687
   DROP EXTERNLOGIN statement, 758
   DROP SYNCHRONIZATION SUBSCRIPTION statement, 778
   GRANT CONSOLIDATE statement [SQL Remote], 834
   MESSAGE statement, 900
   PASSTHROUGH statement [SQL Remote], 913
   START SUBSCRIPTION statement [SQL Remote], 1000
   STOP SUBSCRIPTION statement [SQL Remote], 1008
   SYNCHRONIZE SUBSCRIPTION statement [SQL Remote], 1016
   UNLOAD statement, 1028
TO_CHAR function
   syntax, 386
TO_NCHAR function
   syntax, 387
TODATETIMEOFFSET function
   syntax, 388
TODAY function
syntax, 389
tolerance
  altering tolerance for spatial calculations, 479
  setting tolerance for spatial calculations, 678
TOLERANCE clause
  ALTER SPATIAL REFERENCE SYSTEM
  statement, 479
  CREATE SPATIAL REFERENCE SYSTEM
  statement, 678
TOP clause
  DELETE statement, 742
  SELECT statement, 957
  UPDATE statement, 1039
TRACEBACK function
  syntax, 389
TRACED_PLAN function
  syntax, 390
tracing
  ATTACH TRACING statement, 514
  DETACH TRACING statement, 750
  REFRESH TRACING LEVEL statement, 930
tracing data
  saving using sa_save_trace_data system procedure, 1217
tracing levels
  setting the sa_set_tracing_level system procedure, 1245
trademark information
  retrieving, 1333
Transact-SQL
  alphabetical list of SQL Anywhere Server
  statements, 424
  ANSI equivalency, 348
  bitwise operators, 19
  CASE statement [T-SQL], 534
  catalog procedures, 1091
  comparison operators, 9
  constants, 39
  CONTINUE statement syntax, 1050
  converting SELECT statements, 406
  converting stored procedures, 406
  CREATE FUNCTION statement, 599
  CREATE MESSAGE statement, 614
  CREATE PROCEDURE statement, 637
  CREATE SCHEMA statement syntax, 654
  CREATE TABLE statement syntax, 702
  CREATE TRIGGER statement syntax, 720
datetime compatibility, 136
DECLARE CURSOR statement syntax, 732
domains, 131
EXECUTE statement syntax, 796
GOTO statement syntax, 827
IF statement syntax, 852
local variables, 80
money data types, 106
outer join operators, 20
PRINT statement syntax, 918
quoted_identifier option, 39
READTEXT statement syntax, 924
SELECT statement, 965
SET OPTION statement syntax, 987
SET statement syntax, 987
SQL expression compatibility, 39
statement indicators, 424
strings, 39
system and catalog procedures, 1089
system functions, 154
system procedures, 1090
time compatibility, 136
user-defined data types, 131
WHILE statement syntax, 1050
Writetext statement syntax, 1054
Transact-SQL compatibility
  global variables, 82
  views, 1449
Transact-SQL statements
  BEGIN TRANSACTION syntax, 527
  ROLLBACK TRANSACTION syntax, 951
  SAVE TRANSACTION syntax, 953
Transact-SQL string to date/time conversions
  about, 136
  setting with Transact-SQL SET statement, 987
transaction log
  allocating space using ALTER DBSPACE, 432
  backing up using the BACKUP statement, 516
  decrypting using CREATE DECRYPTED
  DATABASE statement, 558
  decrypting using CREATE DECRYPTED FILE
  statement, 560
determining available space, 1133
  encrypting using the CREATE ENCRYPTED
  DATABASE statement, 564
  encrypting using the CREATE ENCRYPTED
  FILE statement, 567
  renaming without backup, 517
TRUNCATE statement, 1019
TRUNCATE privilege clause
  GRANT statement, 830
TRUNCATE statement
  syntax, 1019
TRUNCATE TABLE statement
  syntax, 1019
TRUNCATE TEXT INDEX statement
  syntax, 1021
truncating
  fails if the table has an immediate text index, 1019
  fails if the table is referenced by an immediate
  views, 1019
  tables, 1019
  text indexes, 1021
TRUNCNUM function
  syntax, 394
TRY statement
  syntax, 1022
TSEQUAL function
  syntax, 395
TYPE clause
  ALTER SERVICE statement [HTTP web service], 466
  ALTER SERVICE statement [SOAP web service], 471
  ALTER SPATIAL REFERENCE SYSTEM
  statement, 479
  ALTER SYNCHRONIZATION SUBSCRIPTION
  statement [MobiLink] , 483
  ALTER SYNCHRONIZATION USER statement
  [MobiLink] , 485
  CREATE EVENT statement, 571
  CREATE FUNCTION statement [web clients], 588
  CREATE PROCEDURE statement [web clients], 630
  CREATE SERVICE statement [HTTP web
  service], 466, 663
  CREATE SERVICE statement [SOAP web
  service], 472, 669
  CREATE SPATIAL REFERENCE SYSTEM
  statement, 676
  CREATE SPATIAL UNIT OF MEASURE
  statement, 681
  CREATE SYNCHRONIZATION
  SUBSCRIPTION statement [MobiLink], 687
  CREATE SYNCHRONIZATION USER, 689
GET DESCRIPTOR statement [ESQL], 824
GRANT CONSOLIDATE statement [SQL
Remote], 834
GRANT REMOTE statement [SQL Remote], 844
MESSAGE statement, 899
SET DESCRIPTOR statement [ESQL], 967
type conversion
  about, 137
type expressions
  search conditions, 40
TYPE PLANAR clause
  ALTER SPATIAL REFERENCE SYSTEM
  statement, 479
  CREATE SPATIAL REFERENCE SYSTEM
  statement, 676
TYPE ROUND EARTH clause
  ALTER SPATIAL REFERENCE SYSTEM
  statement, 479
  CREATE SPATIAL REFERENCE SYSTEM
  statement, 676
U
UCASE function
  syntax, 396
UDF
  user defined function, defined, 148
UDP packets
  sending, 1218
UltraLite
  return NULL if you specify NULL argument, 143
UNBOUNDED keyword
  FOLLOWING clause of WINDOW clause, 1053
  PRECEDING clause of WINDOW clause, 1052
UNCONDITIONALLY clause
  STOP DATABASE statement, 1004
  STOP SERVER statement, 1008
undoing
  changes by rolling back transactions, 950
Unicode
  escape sequences, 398
  UNICODE function, 397
  UNISTR function, 398
Unicode data
  storage, 89
Unicode data types
  about, 89
UNICODE function
INDEX

syntax, 397
UNION statement
setting database options, 1026
syntax, 1025
unions
multiple SELECT statements, 1025
unique
constraint in CREATE TABLE statement, 697
UNIQUE clause
ALTER TABLE statement, 491
CREATE INDEX statement, 600
CREATE TABLE statement, 699
DECLARE CURSOR statement, 730
unique indexes
about, 600
UNIQUEIDENTIFIER data type
syntax, 126
UNIQUEIDENTIFIERSTR data type
syntax, 94
UNISTR function
syntax, 398
units of measure
adding comments using the COMMENT statement, 538
ALTER SPATIAL REFERENCE SYSTEM statement, 477
CREATE SPATIAL REFERENCE SYSTEM statement, 673
CREATE SPATIAL UNITS OF MEASURE statement, 681
installing predefined, 1164
ST_UNITS_OF_MEASURE system view, 1423
universally unique identifiers
SQL syntax for NEWID function, 303
Unix
compressing strings, 181
decompressing strings, 219
UNKNOWN conditions
IS UNKNOWN search condition, 63
UNLIMITED clause
CREATE LOGIN POLICY statement, 607
UNLOAD MATERIALIZED VIEW statement
syntax, 1027
UNLOAD statement
syntax, 1027
UNLOAD TABLE statement
syntax, 1027
unloading
cost models, 1262
data using the UNLOAD statement, 1027
result sets using the UNLOAD statement, 1027
unloading data
multibyte character sets, 1029
UNNEST operator
syntax, 18
UNSIGNED BIGINT data type
syntax, 97
UNSIGNED INTEGER data type
syntax, 102
UNSIGNED SMALLINT data type
syntax, 105
UNSIGNED TINYINT data type
syntax, 105
unzip utility
DECOMPRESS function, 219
UPDATE clause
CREATE PUBLICATION statement [MobiLink]
[SQL Remote], 647
CREATE TRIGGER statement, 714
INSTALL EXTERNAL OBJECT statement, 866
update column privilege
SYSCOLPERM system view, 1351
UPDATE privilege
GRANT statement, 828
REVOKE statement, 946
UPDATE privilege clause
GRANT statement, 830
UPDATE SET clause
MERGE statement, 896
UPDATE statement
(positioned) statement syntax, 1032
setting database options, 1040
SQL Remote syntax, 1034
syntax, 1037
updates
based on joins, 1040
SQL Remote joins, 1035
system tables, 1055
system views, 1347
updating
columns without logging, 1054
publications and subscriptions, 1040
rows, 1037
SQL Remote tables and columns, 1034
UPDLOCK table hint
FROM clause, 816

upgrading databases
  ALTER DATABASE statement, 426
UPPER function
  syntax, 399
uppercase characters
  UPPER function, 399
uppercase strings
  UCASE function, 396
  UPPER function, 399
URI redirection
  CREATE PROCEDURE statement [web clients], 632
URL clause
  ALTER LDAP SERVER statement, 443
  ALTER SERVICE statement [HTTP web service], 466
  CREATE FUNCTION statement [web clients], 588
  CREATE LDAP SERVER statement, 604
  CREATE PROCEDURE statement [web clients], 629
  CREATE SERVICE statement [HTTP web service], 467, 663
USE DEFAULT clause
  CREATE TABLE statement, 690
USER
  CONNECT statement, 543
USER clause
  ALTER SERVICE statement [HTTP web service], 466
  ALTER SERVICE statement [SOAP web service], 471
  CREATE SERVICE statement [HTTP web service], 469, 666
  CREATE SERVICE statement [SOAP web service], 475, 672
  CREATE TABLE statement, 690
user defined functions
  about, 148
  defined, 148
  exiting from, 940
  Java, 148
user defined properties
  incrementing values, 1263
  setting values, 1265
user estimates
  about, 64
user IDs
  restrictions, 833
  revoking, 944
  system views, 1399
  views, 1446
USER special value
  syntax, 77
USER TYPES clause
  DESCRIBE statement, 744
User-Agent
  HTTP_HEADER function, 266
  HTTP_RESPONSE_HEADER function, 268
user-defined data types
  about, 130
  CREATE DOMAIN statement, 562
  dropping using the DROP DATATYPE statement, 755
  Transact-SQL, 131
user-defined functions
  CREATE FUNCTION statement, 594
  returning values from, 940
user-defined options
  temporary settings unsupported, 972
user-defined roles
  granting using the GRANT statement, 828
  revoking using the REVOKE statement, 945
user-extended roles
  converting user-extended roles back to users, 769
  creating, CREATE ROLE statement, 652
user-supplied selectivity estimates
  about, 64
user_estimates option
  overriding in a MERGE statement, 897
  setting for DELETE statements, 743
  setting for EXCEPT statement, 790
  setting for INSERT statements, 863
  setting for INTERSECT statement, 870
  setting for UNION statement, 1026
  setting for UPDATE statements, 1040
USER_ID function
  syntax, 400
USER_NAME function
  syntax, 400
userid
  common element in SQL syntax, 422
users
  adding comments using the COMMENT statement, 538
ALTERN SYCHRONIZATION USER statement, 485
altering using ALTER USER statement, 508
CREATE SYCHRONIZATION USER statement, 688
creating using CREATE USER statement, 721
DROP SYCHRONIZATION USER statement, 779
dropping, 944
dropping using DROP USER statement, 786
obtaining status, 1152
setting, 990
USING AUTO PARENT clause
ALTER MIRROR SERVER statement, 454
CREATE MIRROR SERVER statement, 617
USING clause
ALTER PUBLICATION statement, 457
ALTER SERVER statement, 464
CREATE PUBLICATION statement [MobiLink] [SQL Remote], 647
CREATE SERVER statement, 658
DECLARE CURSOR statement, 731
EXECUTE statement, 794
FOR statement, 805
INPUT statement, 858
MERGE statement, 894
OUTPUT statement, 910
USING CLIENT FILE clause
LOAD TABLE statement, 875
USING COLUMN clause
LOAD TABLE statement, 876
USING DESCRIPTOR clause
EXPLAIN statement [ESQL], 799
GET DATA statement, 822
GET OPTION statement [ESQL], 825
OPEN statement, 904
PUT statement [ESQL], 919
UPDATE (positioned) statement, 1033
USING FILE clause
LOAD TABLE statement, 875
USING VALUE clause
LOAD TABLE statement, 876
UTC TIMESTAMP clause
CREATE TABLE statement, 690
UTC TIMESTAMP default
CREATE TABLE statement, 489, 695
UTC TIMESTAMP special value
syntax, 78
UTF-16 encoding
CSCONVERT function, 198
LOAD TABLE statement, 877, 879
UNLOAD statement, 1028
UUIDs
SQL syntax for NEWID function, 303
SQL syntax for STRTOUUID function, 377
SQL syntax for UUIDTOSTR function, 401
UNIQUEIDENTIFIER data type, 126
UUIDTOSTR function
syntax, 401
VALIDATE CHECKSUM statement
syntax, 1045
VALIDATE DATABASE statement
syntax, 1045
VALIDATE INDEX statement
syntax, 1045
VALIDATE LDAP SERVER statement
syntax, 1044
VALIDATE MATERIALIZED VIEW statement
syntax, 1045
VALIDATE statement
syntax, 1045
VALIDATE TABLE statement
syntax, 1045
VALIDATE TEXT INDEX statement
syntax, 1045
validating
checksums, 1045
indexes using VALIDATE statement, 1045
LDAP server configuration objects, 1044
sa_validate system procedure, 1266
tables using VALIDATE TABLE statement, 1045
VALIDATE statement, 1045
VALUE clause
GET DESCRIPTOR statement [ESQL], 824
SET DESCRIPTOR statement [ESQL], 967
values
returning from procedures, 940
VALUES clause
INSERT statement, 861
VAR_POP function
syntax, 402
VAR_SAMP function
syntax, 404
VARBINARY data type
  syntax, 127
VARBIT data type
  syntax, 108
VARCHAR data type
  byte-length semantics, 95
  character-length semantics, 95
  syntax, 95
  using DESCRIBE on a VARCHAR column, 95
VAREXISTS function
  syntax, 406
variable result sets
  from procedures, CREATE PROCEDURE statement, 641
  from procedures, CREATE PROCEDURE statement [External call], 622
  from procedures, DESCRIBE statement [ESQL], 746
  from procedures, PREPARE statement, 916
variable-name
  common element in SQL syntax, 422
variables
  connection-level variables, 81
  creating SQL, 722
  declaring using DECLARE statement, 736
  dropping SQL variables using the DROP VARIABLE statement, 787
  getting from within a descriptor area, 824
  global variables, 82
  initial value, 737
  local variables, 80
  setting values, 985
  syntax, 79
  using in view definitions, 725
VARIANCE function
  syntax, 406
VERBOSE clause
  OUTPUT statement, 910
VERIFY clause
  UPDATE statement, 1037
  UPDATE statement [SQL Remote], 1034
verifying
  passwords, 1267
version number
  retrieving, 1333
view dependencies
  unloading/reloading databases, 1209
viewing
  Interactive SQL procedure profiling data, 1206
  ViewLastRefreshed property
    sa_materialized_view_info system procedure, 1176
  ViewName property
    sa_materialized_view_info system procedure, 1176
views
  about, 1347
  adding comments using the COMMENT statement, 538
  altering materialized views owned by someone else, 512
  altering using the ALTER VIEW statement, 511
  compatibility views, 1438
  consolidated views, 1419
  CREATE MATERIALIZED VIEW statement, 612
  CREATE VIEW statement, 724
  determining dependencies, 1123
  DROP VIEW statement, 788
  indexes, 602
  parameterized views, 725
  sa_recompile_views system procedure, 1209
  system views, 1348
  Transact-SQL compatibility, 1449
  updating using INSERT statement, 864
VIRTUAL clause
  CREATE INDEX statement, 600
VM
  START JAVA statement, 998
  STOP JAVA statement, 1006
W
WAIT AFTER END clause
  BACKUP statement, 517
WAIT BEFORE START clause
  BACKUP statement, 517
WAITFOR statement
  syntax, 1047
watch list
  configuring with sa_server_option, 1222
Watcom SQL
  DECLARE statement, 736
Watcom SQL statements
  rewriting to Transact-SQL, 391
WATCOMSQL function
  syntax, 406
web clients
DENSE_RANK function, 222
MAX function, 292
MEDIAN function, 293
MIN function, 295
PERCENT_RANK function, 316
RANK function, 326
REGR_AVGX function, 331
REGR_AVGY function, 332
REGR_COUNT function, 334
REGR_INTERCEPT function, 335
REGR_R2 function, 337
REGR_SLOPE function, 338
REGR_SXX function, 339
REGR_SXY function, 341
ROW_NUMBER function, 353
STDDEV function, 372
STDDEV_POP function, 372
STDDEV_SAMP function, 373
SUM function, 381
VAR_POP function, 402
VAR_SAMP function, 404
window-name
common element in SQL syntax, 422
window-spec
syntax in window functions, 1051
windows (OLAP)
WINDOW clause, 1051
WITH ACTIVATE clause
ALTER LDAP SERVER statement, 444
CREATE LDAP SERVER statement, 605
WITH ADMIN ONLY OPTION clause
not applicable for system roles, 829
WITH ADMIN OPTION clause
not applicable for system roles, 829
WITH AUTO NAME clause
INSERT statement, 861
MERGE statement, 895
WITH BATCH clause
EXECUTE IMMEDIATE statement, 792
WITH CHECK OPTION clause
ALTER VIEW statement, 511
CREATE VIEW statement, 725
WITH CHECKPOINT clause
LOAD TABLE statement, 883
WITH CHECKPOINT LOG clause
BACKUP statement, 519
WITH clause
ALTER LDAP SERVER statement, 444
INTERSECT statement, 870
REFRESH MATERIALIZED VIEW statement, 925
REFRESH TEXT INDEX statement, 928
SELECT statement, 957
WITH COMMENT clause
BACKUP statement, 518
WITH CONTENT LOGGING clause
LOAD TABLE statement, 884
required for database mirroring, 886
WITH DROP ALL REFERENCES clause
DROP LDAP SERVER statement, 760
WITH ESCAPES clause
EXECUTE IMMEDIATE statement, 792
WITH EXCLUSIVE MODE clause
REFRESH MATERIALIZED VIEW statement, 925
REFRESH TEXT INDEX statement, 928
WITH EXECUTE clause
PREPARE statement, 915
WITH EXPRESS CHECK clause
VALIDATE statement, 1045
WITH FILE NAME LOGGING clause
LOAD TABLE statement, 884
WITH HOLD clause
LOCK TABLE statement, 890
OPEN statement, 904
WITH HOLD cursors
DECLARE CURSOR statement [ESQL] [SP], 729
locking, 904
OPEN statement, 904
WITH ISOLATION LEVEL clause
REFRESH MATERIALIZED VIEW statement, 925
REFRESH TEXT INDEX statement, 928
WITH LOG clause
WRITETEXT statement [T-SQL], 1054
WITH MAX clause
ALLOCATE DESCRIPTOR statement [ESQL], 425
WITH NO ADMIN OPTION clause
not applicable for system roles, 829
WITH NULLS NOT DISTINCT clause
CREATE INDEX statement, 601
WITH OPTION clause
SETUSER statement, 990
WITH QUOTES clause
EXECUTE IMMEDIATE statement, 791
WITH RECOMPILE clause
CREATE PROCEDURE statement [T-SQL], 637
WITH RECURSIVE clause
SELECT statement, 957
WITH REFRESH clause
ALTER LDAP SERVER statement, 444
WITH RESULT SET clause
EXECUTE IMMEDIATE statement, 792
WITH ROW LOGGING clause
LOAD TABLE statement, 884
required for database mirroring, 886
WITH SAVE clause
DETACH TRACING statement, 750
WITH SCRIPTED UPLOAD clause
CREATE PUBLICATION statement [MobiLink] [SQL Remote], 647
WITH SERVER NAME clause
START DATABASE statement, 994
WITH SHARE MODE clause
REFRESH MATERIALIZED VIEW statement, 925
REFRESH TEXT INDEX statement, 928
WITH SUSPEND clause
ALTER LDAP SERVER statement, 444
DROP LDAP SERVER statement, 761
WITH TEXTPTR clause
GET DATA statement, 823
WITH TRUNCATE AT CHECKPOINT clause
START DATABASE statement, 993
WITH VARIABLE RESULT clause
DESCRIBE statement [ESQL], 746
PREPARE statement, 916
WITHOUT SAVE clause
DETACH TRACING statement, 750
WORK clause
COMMIT statement, 541
write checksums
validating, 1045
WRITE_CLIENT_FILE function
syntax, 408
WriteNoPK locks
sa_locks system procedure, 1170
WRITETEXT statement
Transact-SQL syntax, 1054
writing files
using xp_write_file, 1344
WSDL
CREATE FUNCTION statement [web clients], 590
CREATE PROCEDURE statement [web clients], 635
CREATE SERVICE statement [SOAP web service], 472, 669
WWW-Authenticate
HTTP_RESPONSE_HEADER function, 268
X
XLOCK table hint
FROM clause, 816
XML
CREATE SERVICE statement [HTTP web service], 466, 663
loading from, using the LOAD TABLE statement, 881
OPENXML operator, 12
XML data type, 96
XMLAGG function, 410
XMLCONCAT function, 411
XMLELEMENT function, 412
XMLFOREST function, 414
XMLGEN function, 415
XML data type
syntax, 96
XMLAGG function
syntax, 410
XMLATTRIBUTES parameter
XMLELEMENT function, 412
XMLCONCAT function
syntax, 411
XMLELEMENT function
syntax, 412
XMLFOREST function
syntax, 414
XMLGEN function
syntax, 415
xp_cmdshell system procedure
syntax, 1328
xp_get_mail_error_code system procedure
syntax, 1329
xp_get_mail_error_text system procedure
syntax, 1331
xp_getenv system procedure
syntax, 1332
xp_msver system procedure

xp_read_file system procedure
  syntax, 1334
xp_scanf system procedure
  syntax, 1335
xp_sendmail system procedure
  syntax, 1336
xp_sprintf system procedure
  syntax, 1339
xp_startmail system procedure
  syntax, 1339
xp_startsmtp system procedure
  enabling in McAfee VirusScan, 1343
  possible conflicts with virus scanner settings, 1343
  syntax, 1341
xp_stopmail system procedure
  syntax, 1343
xp_stopsmtplib system procedure
  syntax, 1344
xp_write_file system procedure
  syntax, 1344

Y
YEAR function
  syntax, 417
YEARS function
  syntax, 417
YMD function
  syntax, 419

Z
ZIP file
  email example, 1338
zip utility
  COMPRESS function, 181