Contents

About this book ........................................................................................................................... vii

UltraLite overview ........................................................................................................................ 1
  UltraLite architecture ..................................................................................................................... 1
  UltraLite synchronization client features ..................................................................................... 2
  UltraLite supported platforms ....................................................................................................... 3
  UltraLite, UltraLite Java edition, and SQL Anywhere feature comparisons .................................. 3
  UltraLite and UltraLite Java edition database limitations ............................................................ 8
  CustDB sample application overview .......................................................................................... 12
  UltraLite solution considerations for Windows Mobile ................................................................. 15

UltraLite database creation approaches ..................................................................................... 19
  Creating an UltraLite database with the Create Database Wizard ................................................. 19
  UltraLite database creation using a command prompt .................................................................... 20
  UltraLite database creation using a MobiLink synchronization model ........................................ 20
  UltraLite database creation through central administration of remote databases .......................... 21
  Creating an UltraLite database from an XML file ......................................................................... 21
  UltraLite database creation on a first connection .......................................................................... 22
  UltraLite database creation parameters ....................................................................................... 23
  UltraLite character sets ............................................................................................................... 24
  Database security ......................................................................................................................... 26

Conversion from a SQL Anywhere database to an UltraLite database ........................................ 29

UltraLite database connections .................................................................................................... 31
  UltraLite connection strings and parameters ............................................................................... 31
  UltraLite connection parameters and the ULSQLCONNECT environment variable .......................... 32
# UltraLite® - Database Management and Reference

**UltraLite file path formats in connection parameters** .................................................. 33

**UltraLite database tasks and features** ................................................................. 35
- Reading database properties ........................................................................ 35
- Accessing database options ........................................................................ 35
- UltraLite event notifications ........................................................................ 36
- Isolation levels ............................................................................................... 39
- Validating an UltraLite database .................................................................. 41
- UltraLite and UltraLite Java edition database back up and recovery .......... 42

**UltraLite database schemas** .............................................................................. 45
- UltraLite tables and columns ......................................................................... 46
- UltraLite indexes ............................................................................................ 52
- UltraLite users ............................................................................................... 56

**UltraLite as a MobiLink client** ........................................................................ 63
- UltraLite clients ............................................................................................ 63
- ActiveSync with UltraLite on Windows Mobile ............................................ 81
- UltraLite synchronization parameters ......................................................... 82
- UltraLite network protocol options ............................................................... 104

**UltraLite deployment** ...................................................................................... 107
- UltraLite application build and deployment specifications ......................... 107
- UltraLite and UltraLite Java edition database deployment techniques ..... 115
- Deploying UltraLite database schema upgrades .......................................... 115
- UltraLite engine startup .............................................................................. 117
- Deploying the ActiveSync provider for UltraLite ...................................... 118
- Registering applications with the ActiveSync Manager ......................... 120

**Tutorial: Building the UltraLite CustDB sample application** ........... 123
- Lesson 1: Building and running the CustDB application ......................... 123
Unable to start the UltraLite engine ............................................................... 461
Unable to connect to databases after upgrade ............................................. 461
UltraLite database corruption ......................................................................... 462
Database size not stabilizing ........................................................................ 463
Importing ASCII data into a new UltraLite database ................................. 463
Utilities still running as the previous version ............................................ 464
Result set changes unpredictably ................................................................. 464
UltraLite engine client fails with error -764 ............................................... 465

Index ...................................................................................................  467
About this book

This book introduces the UltraLite database systems for small devices.
UltraLite overview

UltraLite is a compact relational database management system with many of the same features as SQL Anywhere. It can be used to create mobile databases for small-footprint devices such as smartphones, handheld computers, and tablet PCs.

UltraLite includes a built-in synchronization client that tracks changes made in UltraLite and UltraLite Java edition databases, and exchanges updates with a MobiLink server over a network. As a MobiLink client, UltraLite ensures that mobile applications can stay synchronized with a central database and with other UltraLite databases.

UltraLite architecture

UltraLite supports a multitude of mobile platforms and consists of API development, database management, and database layers, as represented in the following diagram:

- **Mobile platform support** Your target mobile platform determines which UltraLite API is available for application development. For more information about supported platforms and devices, see [http://www.sybase.com/detail?id=1061806](http://www.sybase.com/detail?id=1061806).

- **API development layer** Refer to the diagram above to determine which API to use for your target mobile platform. For more information about API development, see:
  - “UltraLite C++ application development” [UltraLite - C and C++ Programming]
  - “UltraLite.NET application development” [UltraLite - .NET Programming]
  - “UltraLiteJ application development” [UltraLite - Java Programming]
• **Database management layer and synchronization client**  
   UltraLite provides two data management systems—the native **UltraLite database management system**, and the **UltraLite Java edition**.
   
   ○ **UltraLite database management system**  
     Most mobile platforms use the UltraLite APIs to interface with the UltraLite database management system. This system allows you to create and connect to an **UltraLite database**.
     
     A comprehensive set of administration tools is provided to help you maintain your UltraLite project. You can run these tools as either command line utilities or wizards in the UltraLite plug-in for Sybase Central. For more information, see “UltraLite utilities” on page 186.
   
   ○ **UltraLite Java edition**  
     The BlackBerry OS platform requires use of the UltraLiteJ API to interface with the UltraLite Java edition to create and connect to an **UltraLite Java edition database**. These databases are not interchangeable with UltraLite databases and have some restrictions. For more information, see “UltraLite and UltraLite Java edition database limitations” on page 8.
     
     UltraLite provides a set of Java edition utilities to perform additional tasks on UltraLite Java edition databases. For more information, see “UltraLite Java edition utilities” on page 234.

   For more information about the UltraLite synchronization client, see “UltraLite synchronization client features” on page 2.

• **Database layer**  
   This layer is the local data repository stored as a file. UltraLite databases are stored as UDB files, and UltraLite Java edition databases are stored as JDB files. UDB files are portable across all mobile platforms except for BlackBerry OS.

---

**UltraLite synchronization client features**

UltraLite includes a built-in bi-directional synchronization client that causes all data in an UltraLite database to be synchronized by default. Users new to MobiLink synchronization may use this default behavior until business requirements necessitate a custom synchronization design to alter what UltraLite data gets synchronized to the consolidated database. Unlike SQL Anywhere remote databases, you do not need to increase the size of the UltraLite footprint to include synchronization functionality.

Important synchronization features built into the UltraLite runtime include a row state tracking mechanism and a synchronization state tracking mechanism.

**The row state tracking mechanism**

Tracking the state of tables and rows is particularly important for data synchronization. Each row in an UltraLite database has an associated row state structure. In addition to synchronization, UltraLite also uses the row states to control transaction processing and data recovery.

**Synchronization state tracking**

UltraLite uses a progress counter to ensure robust synchronization. Each upload is given a unique number to identify it. This allows UltraLite to determine whether an upload was successful when a communication error occurs.
When you first create a new database, UltraLite always sets the synchronization progress counter to zero. A progress counter value of zero identifies the database as a new UltraLite database, which tells the MobiLink server to reset its state information for this client.

**Caution**
Because UltraLite increments the progress counter each time a synchronization occurs, you cannot synchronize an UltraLite database to different consolidated databases. If the progress counter value is not zero and does not match that sequence number stored in the consolidated database, MobiLink synchronization reports an offset mismatch and synchronization fails. You cannot replace an UltraLite database with a backup copy if the progress counter is older than the current value.

See also
- “UltraLite database row state management” on page 453
- “MobiLink synchronization” [MobiLink - Getting Started]
- “UltraLite as a MobiLink client” on page 63
- “Tutorial: Using MobiLink with a SQL Anywhere consolidated database” [MobiLink - Getting Started]

**UltraLite supported platforms**
For more information about the devices, platforms, and network protocols supported by UltraLite, see http://www.sybase.com/detail?id=1061806.

See also
- “UltraLite as a MobiLink client” on page 63
- “UltraLite network protocol options” on page 104
- “UltraLite Synchronization utility (ulsync)” on page 210

**UltraLite, UltraLite Java edition, and SQL Anywhere feature comparisons**
The following table illustrates feature comparisons between UltraLite and SQL Anywhere. UltraLite features are supported by UltraLite Java edition unless otherwise specified.

**Note**
The UltraLite database management system adds 750-1500 KB to the size of your application while UltraLite Java edition adds 500 KB. The SQL Anywhere database, database server, and synchronization client add approximately 6 MB.

<table>
<thead>
<tr>
<th>Feature</th>
<th>SQL Anywhere</th>
<th>UltraLite</th>
<th>Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction processing, and multi-table joins</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>SQL Anywhere</td>
<td>UltraLite</td>
<td>Considerations</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Triggers, stored procedures, and views</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>External stored procedures (callable external DLLs)</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Built-in referential and entity integrity</td>
<td>X</td>
<td>X¹</td>
<td>UltraLite Java edition does not enforce foreign key constraints. See “Avoiding synchronization issues with foreign key cycles” on page 73.</td>
</tr>
<tr>
<td>Cascading updates and deletes</td>
<td>X</td>
<td>Limited¹</td>
<td>Declarative referential integrity, where deletes and updates are cascaded, is a feature that is not supported in UltraLite databases, except during synchronization when deletes are cascaded for this purpose.</td>
</tr>
<tr>
<td>Dynamic, multiple database support</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Multi-threaded application support</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Row-level locking</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>SQL Anywhere</td>
<td>UltraLite</td>
<td>Considerations</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XML unload and load utilities</td>
<td>X</td>
<td></td>
<td>UltraLite uses separate administration tools to complete XML load and unloads. These features are not built into the runtime. For UltraLite, see:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• “UltraLite Load XML to Database utility (ulload)” on page 206</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• “UltraLite Database Unload utility (ulunload)” on page 216</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For UltraLite Java edition, see:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• “UltraLite Java Edition Database Load utility (uljload)” on page 235</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• “UltraLite Java Edition Database Unload utility (uljunload)” on page 237</td>
</tr>
<tr>
<td>SQLX functionality</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL functions</td>
<td>X</td>
<td>X</td>
<td>Not all SQL functions are available for use in UltraLite applications. If you use an unsupported function, you trigger an error. See “UltraLite SQL functions” on page 294.</td>
</tr>
<tr>
<td>SQL statements</td>
<td>X</td>
<td>X</td>
<td>The scope of SQL statements is different compared to SQL Anywhere. See “UltraLite SQL statements” on page 395.</td>
</tr>
<tr>
<td>Integrated HTTP server</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strong encryption for</td>
<td>X</td>
<td>X</td>
<td>An UltraLite event model differs from SQL Anywhere.</td>
</tr>
<tr>
<td>database files and network</td>
<td></td>
<td></td>
<td>communications</td>
</tr>
<tr>
<td>Event scheduling and</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>handling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>SQL Anywhere</td>
<td>UltraLite</td>
<td>Considerations</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>--------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>High-performance, self-tuning, cost-based query optimizer</td>
<td>X</td>
<td></td>
<td>UltraLite has a query optimizer that is not as extensive as that of SQL Anywhere.</td>
</tr>
<tr>
<td>Choice of several thread-safe APIs</td>
<td>X</td>
<td>X</td>
<td>UltraLite gives application developers a uniquely flexible architecture that allows for the creation of applications for changing and/or varied deployment environments. See “Benefits of UltraLite APIs for Windows Mobile” on page 16.</td>
</tr>
<tr>
<td>Cursor support</td>
<td>X</td>
<td>X</td>
<td>See “UltraLite and UltraLite Java edition database limitations”.</td>
</tr>
<tr>
<td>Dynamic cache sizing</td>
<td>X</td>
<td>X¹</td>
<td>UltraLite allows you to set an initial, minimum, and/or maximum cache size for a database. The size of the cache is optimized by UltraLite on an ongoing basis, up to the maximum size (if specified). See:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● “UltraLite CACHE_SIZE connection parameter” on page 160</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● “UltraLite CACHE_MIN_SIZE connection parameter” on page 159</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● “UltraLite CACHE_MAX_SIZE connection parameter” on page 158</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UltraLite Java edition databases support fixed cache sizes only.</td>
</tr>
<tr>
<td>Database recovery after system or application failure</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Binary Large Object (BLOB) support</td>
<td>X</td>
<td>X</td>
<td>UltraLite cannot index or compare BLOBs.</td>
</tr>
<tr>
<td>Windows Performance Monitor integration</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Online table and index defragmentation</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Feature</strong></td>
<td><strong>SQL Anywhere</strong></td>
<td><strong>UltraLite</strong></td>
<td><strong>Considerations</strong></td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Online backup</td>
<td>X</td>
<td></td>
<td>SQL Anywhere databases need a database server before allowing desktop connections to the database that you deploy on a Windows Mobile device. On UltraLite, you prefix the connection string with <code>WCE:\</code>. See “Windows Mobile” on page 34.</td>
</tr>
<tr>
<td>Direct device connections to a Windows Mobile device from the desktop.</td>
<td></td>
<td>X(^1)</td>
<td></td>
</tr>
<tr>
<td>High-performance updates and retrievals through the use of indexes</td>
<td>X</td>
<td>X</td>
<td>UltraLite uses a mechanism to determine whether each table is searched using an index or by scanning the rows directly. Additionally, you can hash indexes to speed up data retrieval. See “UltraLite max_hash_size creation parameter” on page 141.</td>
</tr>
<tr>
<td>Synchronizing to HANA, Oracle, DB2, Sybase Adaptive Server Enterprise, Microsoft SQL Server, MySQL, or SQL Anywhere</td>
<td>X</td>
<td>X</td>
<td>Unlike SQL Anywhere deployments, UltraLite does not require a client agent for synchronization. Synchronization is built into the UltraLite runtime to minimize the components you need to deploy. See “UltraLite clients” on page 63.</td>
</tr>
<tr>
<td>Built-in synchronization</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>In-process execution</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Computed columns</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Declared temporary tables/global temporary tables</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System functions</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Feature Considerations

<table>
<thead>
<tr>
<th>Feature</th>
<th>SQL Anywhere</th>
<th>UltraLite</th>
<th>SQL Anywhere supports the DEFAULT TIMESTAMP default. UltraLite only supports the DEFAULT CURRENT TIMESTAMP default. Therefore, UltraLite can not automatically update the timestamp when the row is updated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp columns</td>
<td>X</td>
<td>X</td>
<td><strong>User-based permission scheme to determine object-based ownership and access</strong></td>
</tr>
<tr>
<td>UltraLite is primarily designed for single user databases in which an authorization system is not needed. However, you can include up to four user IDs and passwords, which are used for authentication purposes only. These users have access to all database objects. See “UltraLite users” on page 56.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spatial data</td>
<td>X</td>
<td>Limited</td>
<td>UltraLite supports point data only.</td>
</tr>
<tr>
<td>Full text data</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 Not available for UltraLite Java edition.

## UltraLite and UltraLite Java edition database limitations

The following table illustrates hard limits that apply to UltraLite and UltraLite Java edition databases. In many cases, the limits are beyond the maximum capabilities of mobile devices. Performance considerations and device capabilities impose stricter limitations.

<table>
<thead>
<tr>
<th>Item</th>
<th>UltraLite database limitations</th>
<th>UltraLite Java edition database limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database and file size</td>
<td>4 GB or less if there is an operating system limit on file size.</td>
<td>Limited by the operating system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of database pages is limited to 64 K.</td>
</tr>
<tr>
<td>Temporary file size</td>
<td>Limited by the operating system.</td>
<td>Not applicable.</td>
</tr>
<tr>
<td>Item</td>
<td>UltraLite database limitations</td>
<td>UltraLite Java edition database limitations</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Cache size</td>
<td>Limited by the available memory on the device.</td>
<td>Limited by the available memory on the device.</td>
</tr>
</tbody>
</table>
| Dynamic cache sizing                          | UltraLite allows you to set an initial, minimum, and/or maximum cache size for a database. The size of the cache is optimized by UltraLite on an ongoing basis, up to the maximum size (if specified). See:  
  - “UltraLite CACHE_SIZE connection parameter” on page 160  
  - “UltraLite CACHE_MIN_SIZE connection parameter” on page 159  
  - “UltraLite CACHE_MAX_SIZE connection parameter” on page 158 | UltraLite Java edition databases support fixed cache sizes only.                                                |
<p>| Maximum number of concurrent open connections supported by a database | Up to 14.                                                                                      | No limit.                                                                                                    |
| Maximum number of concurrent open connections to all databases                    | Up to 16 on a mobile device, and up to 64 on a desktop.                                         | No limit.                                                                                                    |
| Maximum number of databases that can run concurrently                        | Up to 8 on a mobile device, and up to 32 on a desktop.                                          | No limit.                                                                                                    |
| Maximum number of applications that can connect to a database concurrently      | Use the UltraLite engine to handle multiple concurrent applications connecting to the database. Otherwise, only one application can connect to a database at one time. | 1                                                                                                            |
| SQL communication areas (SQLCA)                                                | Up to 63.                                                                                      | Not applicable.                                                                                              |</p>
<table>
<thead>
<tr>
<th>Item</th>
<th>UltraLite database limitations</th>
<th>UltraLite Java edition database limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rows per table</td>
<td>Up to 16 million.</td>
<td>Limited by the page size and the maximum number of pages per database.</td>
</tr>
<tr>
<td></td>
<td>Sometimes changes to the row (deletes and updates) and other state information are maintained with the row data. This information allows those changes to be synchronized. So, the actual row limit can be smaller than 16 million, depending on the number of transactions on a table between synchronization, or whether the table is synchronized at all. See “UltraLite transaction processing” on page 454.</td>
<td></td>
</tr>
<tr>
<td>Row Size</td>
<td>The length of each packed row must not exceed the page size. See “Row packing and table definitions” on page 46.</td>
<td>The length of each packed row must not exceed the page size. See “Row packing and table definitions” on page 46.</td>
</tr>
<tr>
<td></td>
<td>Character strings are stored without padding when they are shorter than the column size. This restriction excludes columns declared as LONG BINARY and LONG VARCHAR as these strings are stored separately.</td>
<td>Character strings are stored without padding when they are shorter than the column size. This restriction excludes columns declared as LONG BINARY and LONG VARCHAR as these strings are stored separately.</td>
</tr>
<tr>
<td>Rows per database</td>
<td>Limited by the persistent store.</td>
<td>Limited by the persistent store.</td>
</tr>
<tr>
<td>Table size</td>
<td>Limited by the database size.</td>
<td>Limited by the database size.</td>
</tr>
<tr>
<td>Tables per database</td>
<td>Limited by the database size.</td>
<td>Up to 32000.</td>
</tr>
<tr>
<td>Columns per table</td>
<td>Row size is limited by page size, so the practical limit on the number of columns per table is derived from this size. Typically, this practical limit is much less than 4000.</td>
<td>Row size is limited by page size, so the practical limit on the number of columns per table is derived from this size. Typically, this practical limit is much less than 4000.</td>
</tr>
<tr>
<td>Indexes per table</td>
<td>Limited by the database size.</td>
<td>Limited by the database size.</td>
</tr>
<tr>
<td>Number of publications</td>
<td>Up to 63.</td>
<td>Up to 63.</td>
</tr>
</tbody>
</table>
## UltraLite and UltraLite Java edition database limitations

<table>
<thead>
<tr>
<th>Item</th>
<th>UltraLite database limitations</th>
<th>UltraLite Java edition database limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database page size</td>
<td>Minimum 1 K; up to 16 KB.</td>
<td>Minimum 256 bytes; up to 16 KB.</td>
</tr>
<tr>
<td>Cursors per connection</td>
<td>The maximum number of allowable cursors on a given connection to an UltraLite database is 64 (all platforms).</td>
<td>The maximum number of allowable cursors on a given connection to an UltraLite database is 64 (all platforms).</td>
</tr>
<tr>
<td>Strings</td>
<td>The row must fit on a page.</td>
<td>The row must fit on a page.</td>
</tr>
<tr>
<td>Binary data types</td>
<td>The row must fit on a page.</td>
<td>The row must fit on a page.</td>
</tr>
<tr>
<td>Long binary/long varchar size</td>
<td>Limited only by database size.</td>
<td>Limited only by database size.</td>
</tr>
<tr>
<td>Blob size</td>
<td>Limited by file size.</td>
<td>Up to $2^{24}$ bytes.</td>
</tr>
<tr>
<td>Available Isolation levels</td>
<td>0 (read uncommitted) or 1 (read committed).</td>
<td>0 (read uncommitted).</td>
</tr>
<tr>
<td>Cascading updates and deletes</td>
<td>Declarative referential integrity, where deletes and updates are cascaded, is a feature that is not supported in UltraLite databases, except during synchronization when deletes are cascaded for this purpose.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>Event scheduling and handling</td>
<td>An UltraLite event model differs from SQL Anywhere.</td>
<td>Not supported.</td>
</tr>
</tbody>
</table>
UltraLite database limitations

<table>
<thead>
<tr>
<th>Item</th>
<th>UltraLite database limitations</th>
<th>UltraLite Java edition database limitations</th>
</tr>
</thead>
</table>
| UltraLite Java edition database compatibility | UltraLite databases are not interchangeable with UltraLite Java edition databases. An UltraLite database can be converted into an UltraLite Java edition database and vice versa using the respective load and unload utilities. See:  
  - “UltraLite Database Unload utility (ulunload)” on page 216  
  - “UltraLite Load XML to Database utility (ulload)” on page 206 | UltraLite databases are not interchangeable with UltraLite Java edition databases. An UltraLite database can be converted into an UltraLite Java edition database and vice versa using the respective load and unload utilities. See:  
  - “UltraLite Java Edition Database Unload utility (uljunload)” on page 237  
  - “UltraLite Java Edition Database Load utility (uljload)” on page 235 |

Object store limitations for BlackBerry

On BlackBerry smartphones, the size of the database store is limited by the number of object handles available. The number of available object handles is a BlackBerry feature and is determined by the size of the flash memory.

UltraLite requires one or more object handles, depending on the handle type, to store a database value in memory. For example, a table row with ten columns and two indexes requires a minimum of twelve object handles.

To permit larger database stores, the UltraLiteJ API allows users to limit the number of rows kept in memory. Stored database rows are combined on a database page. The UltraLiteJ API requires only one persistent object handle for each database page.

See also

- “SQL Anywhere size and number limitations” [SQL Anywhere Server - Database Administration]

CustDB sample application overview

The CustDB sample is installed with SQL Anywhere. It is a multi-tiered database management solution that implements MobiLink synchronization with a SQL Anywhere consolidated database.

CustDB consists of the following:

- A **consolidated** SQL Anywhere database. The database is pre-populated with sales status data.
- A **remote** UltraLite database. This database is initially empty.
An UltraLite client application.

MobiLink server synchronization scripts.

Note
You can only run one instance of CustDB at a time. Trying to run more than one instance brings the first instance to the foreground.

CustDB allows sales personnel to track and monitor transactions and then pool information from two types of users:

- Sales personnel that authenticate with user IDs 51, 52, and 53.
- Mobile managers that authenticate with user ID 50.

Information gathered by these different users can be synchronized with the consolidated database.

Both the consolidated and remote databases contain a table named ULOrder. While the consolidated database holds all orders (approved and those pending approval), the UltraLite remote database only displays a subset of rows according to the user that has authenticated.

Columns in the table appear as fields in the client application. When you add an order, you must populate the Customer, Product, Quantity, Price, and Discount fields. You can also append other details such as Status or Notes. The timestamp column identifies whether the row needs to be synchronized.

The synchronization logic for CustDB is held in the consolidated database as MobiLink synchronization scripts. Synchronization logic allows you to determine how much of the consolidated database you need to download and/or upload. You can download complete tables or partial tables (with either row or column subsets) using such techniques as timestamp-based synchronization or snapshot synchronization.

You can use Sybase Central to browse the synchronization scripts that are stored in the consolidated database. Sybase Central is the primary tool for adding scripts to the database.

The custdb.sql file adds each synchronization script to the consolidated database by calling ml_add_connection_script or ml_add_table_script. Connection scripts control high-level events that are not associated with a particular table. Use these events to perform global tasks that are required during every synchronization. Table scripts allow actions at specific events relating to the synchronization of a specific table, such as the start or end of uploading rows, resolving conflicts, or selecting rows to download.

SQL Anywhere CustDB database
This is the consolidated database. During installation, an ODBC data source called SQL Anywhere 16 CustDB is created for this database. The database file is located at %SQLANYSAMP16%\UltraLite\CustDB. You can recreate this database at any time using makedbs.cmd (makedbs.sh for Mac OS X or Linux).

You can erase changes that were synchronized into the consolidated CustDB.db file, so you have a clean version to work with using this script: C:\Program Files\SQL Anywhere 16\UltraLite\CustDB\newdb.bat.
For more information about the schema of this file, see “CustDB sample for MobiLink” [MobiLink - Getting Started].

The UltraLite CustDB database

This is the remote version of the consolidated database that contains only a subset of the information, depending on which user synchronizes the database.

The file name and location can vary depending on the platform, programming language, or even device.

- For UltraLite.NET: C:\Program Files\SQL Anywhere 16\UltraLite.NET\ CustDB\Common\custdb.udb
- For all other platforms and APIs: C:\Program Files\SQL Anywhere 16\UltraLite\ CustDB\custdb.udb

The UltraLite database is also recreated by the makedbs.cmd script (makedbs.sh for Mac OS X or Linux).

RDBMS-specific build scripts

The SQL scripts rebuild a CustDB consolidated database for any one of the supported RDBMSs.

In the C:\Program Files\SQL Anywhere 16\MobiLink\ CustDB directory, you can find the following files:

- For SQL Anywhere: custdb.sql
- For Adaptive Server Enterprise: custase.sql
- For Microsoft SQL Server: custmss.sql
- For Oracle: custora.sql
- For IBM DB2: custdb2.sql
- For MySQL: custmys.sql

For more information about setting up a consolidated database, see “CustDB consolidated database setup” [MobiLink - Getting Started].

UltraLite CustDB client applications and ReadMe files

These are the end-user applications that provide a user-friendly interface to the UltraLite remote database. There is a sample client installed for each supported platform.

Each client application also contains a ReadMe.html or ReadMe.txt file. Each file includes an outline of the steps that are required to build and run the sample.

The location of the application and its ReadMe depends on your development environment. See “Lesson 1: Building and running the CustDB application” on page 123.

Synchronization logic

The UltraLite database SQL statements and synchronization calls are located in custdbcpp.cpp for the C++ API and in custdb.sql for embedded SQL.

See also

- “MobiLink consolidated databases” [MobiLink - Server Administration]
CustDB file locations

The CustDB application is built for many development environments.

UltraLite for Windows 32-bit desktop

You do not need to build the CustDB application before running it.

You can find the CustDB executable file in the %SQLANY16%\UltraLite\Windows\x86 directory.

UltraLite for C/C++

- All versions of C/C++  You can find multiple versions of the C/C++ CustDB project file because of the many C/C++ development environments. Most versions use the generic files. These files are located in the C:\Program Files\SQL Anywhere 16\UltraLite\Custdb directory.

  For information about all versions of C/C++ CustDB applications, see C:\Program Files\SQL Anywhere 16\UltraLite\Custdb\readme.txt.

- Visual Studio  You can find project files in the %SQLANYSAMP16%\UltraLite\CustDB directory.

- Xcode for iPhone  For development on Mac OS X and iPhone, see samples/ultralite/custdb/iphone.

UltraLite.NET

You can find project files specific to Microsoft Visual Studio in the C:\Users\Public\Documents\SQL Anywhere 16\Samples\UltraLite.NET\CustDB directory.

UltraLite solution considerations for Windows Mobile

This section outlines the UltraLite design options that are available for Windows mobile development, and the implementation advantages for each option. The following selections are described:

- UltraLite API selection  The benefits of using each of the following APIs for Windows Mobile development are described:
  - UltraLite C/C++
  - UltraLite embedded SQL
  - UltraLite.NET

- Data management component selection  The benefits of using each of the following data management components for Windows Mobile development are described:
  - UltraLite in-process runtime environment
  - UltraLite database engine
Benefits of UltraLite APIs for Windows Mobile

The UltraLite C/C++, embedded SQL, and .NET APIs offer several data access models, including a simple table-based data access interface and dynamic SQL for more complex queries. By combining these benefits, UltraLite gives application developers a flexible architecture for creating applications for their varied deployment environments.

UltraLite.NET API benefits

The UltraLite.NET API is usually recommended for Windows Mobile development because the SQL Anywhere .NET API provides common programming models that are shared between UltraLite components and SQL Anywhere, and because of the .NET programming compared C/C++.

UltraLite C/C++ and embedded SQL API benefits

While UltraLite provides high performance in a variety of environments and use cases, embedded SQL and the UltraLite C/C++ API are the lowest level APIs and generally deliver the highest performance.

Use the UltraLite C/C++ API when you are trying to create the smallest application footprint. These applications typically yield the best performance and still maintain a small application file size.

See also

- “.NET application programming” [SQL Anywhere Server - Programming]
- “UltraLite C++ application development” [UltraLite - C and C++ Programming]
- “UltraLite C++ application development using embedded SQL” [UltraLite - C and C++ Programming]
- “UltraLite.NET application development” [UltraLite - .NET Programming]

UltraLite data management components for Windows Mobile

UltraLite allows you to build a small-footprint relational database solution without requiring the additional overhead of setting up a separate database server. Instead, UltraLite programming interfaces use one of two approaches—the UltraLite in-process runtime library and the UltraLite database engine. Both approaches control connection and data access requests.

Both components include a built-in bi-directional synchronization client that links UltraLite databases with the MobiLink synchronization server. For more information about the UltraLite synchronization client, see “UltraLite synchronization client features” on page 2.

UltraLite in-process runtime library

The UltraLite in-process runtime is recommended when only one application needs to access a database at a time.

The runtime and the application are part of the same process, which makes the database specific to the application. The runtime library manages UltraLite databases and built-in synchronization operations.

Linking to the runtime requires a different import library/DLL pair from that of the engine.
UltraLite supports both static and dynamic linkage.

- **Static linking**  Static linking requires less device memory and is more effective when only a single UltraLite application is used on the device.

- **Dynamic linking**  Dynamic linking may be more economical with device memory when multiple UltraLite applications are used on the device.

**UltraLite database engine (the uleng16.exe utility)**

The UltraLite engine is only available for Windows desktop and Windows Mobile platforms. The engine is a separate executable that supports concurrent access from multiple applications. Each application must use a client library to communicate with the UltraLite engine.

The UltraLite engine requires more system resources than the UltraLite runtime and may yield lower performance when large amounts of data are moved between the client and database.

Connecting to the engine requires that you specify a different import library/DLL pair than that of the runtime.

The UltraLite engine is required under the following conditions:

- Multiple processes access the same database file at potentially the same time (same time means multiple processes have connections open to the same database at the same time).

- Central Administration is used to manage the UltraLite application database.

**See also**

- “UltraLite application build and deployment specifications” on page 107
- “How to build and deploy UltraLite C++ applications” [*UltraLite - C and C++ Programming*]
- “UltraLite engine startup” on page 117
- “UltraLite Engine utility (uleng16)” on page 195
- “UltraLite concurrency” on page 452
- “UltraLite, UltraLite Java edition, and SQL Anywhere feature comparisons” on page 3
- “UltraLite and UltraLite Java edition database limitations” on page 8
UltraLite database creation approaches

There are three common types of database creation methods:

● Desktop creation methods with UltraLite administration tools designed for database creation.

● On-device creation methods with UltraLite APIs.
  
  On-device creation methods are primarily described in each API specific UltraLite programming book.

● A Central Administration remote task, configured to create an UltraLite database on a device.

Once the database is created, you can connect to it and build tables and other database objects.

See also

● “UltraLite database creation parameters” on page 23
● “Accessing database options” on page 35

Creating an UltraLite database with the Create Database Wizard

Create a database using Sybase Central.

Prerequisites

There are no prerequisites for this task.

Context and remarks

Choose this method if you want help navigating the available database creation parameters. This wizard simplifies your choices by restricting what you can configure based on the target platform(s) you select. Once the database is created, it displays the command line syntax that you can record and use with the ulinit utility.

Task

1. Click Start » Programs » SQL Anywhere 16 » Administration Tools » Sybase Central.

2. Click Tools » UltraLite 16 » Create Database.

3. Follow the instructions in the Create Database Wizard.

Results

The database is created.
You can now connect to the database and build tables and other database objects.

See also
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite database creation parameters” on page 23

**UltraLite database creation using a command prompt**

Choose any of the following utilities to create a database at a command prompt:

- Use the ulinit utility to create a new, empty UltraLite database or one sourced from a SQL Anywhere reference database schema. With this utility, you can include utility options to configure the database. See “Conversion from a SQL Anywhere database to an UltraLite database” on page 29.

- Use the ulload utility if you have an XML file that will serve as the source point for the schema and/or data of your new UltraLite database. See “Creating an UltraLite database from an XML file” on page 21.

- Central Administration: Choose this method if you have a deployment where the MobiLink Agent is configured on all your deployed devices, or you are unable to deploy your initial UltraLite database with your application. You can configure a remote task to create a new UltraLite database on the device. This database can then be managed centrally by an administrator. See “Central administration of remote databases” [MobiLink - Server Administration].

**Create a new UltraLite database (command line)**

Run the ulinit utility specifying the new UltraLite database file to accept the defaults:

```
ulinit test.udb
```

For information about the supported options, see “UltraLite Initialize Database utility (ulinit)” on page 198.

**UltraLite database creation using a MobiLink synchronization model**

To simplify development, MobiLink includes a Create Synchronization Model Wizard to create your UltraLite database and server-side synchronization logic.

Choose this method if you are creating a synchronization system with remote UltraLite databases and a centralized consolidated database.
Once you have created your model, you can work in MobiLink Model mode in Sybase Central to customize your synchronization model before you deploy it. When the model is ready, you can then deploy it to generate the scripts and tables required for your synchronization application.

See also

- “MobiLink plug-in for Sybase Central” [MobiLink - Getting Started]

UltraLite database creation through central administration of remote databases

MobiLink provides a create database command that allows you to create an UltraLite database.

See also

- “Create database command” [MobiLink - Server Administration]
- “Central administration of remote databases” [MobiLink - Server Administration]

Creating an UltraLite database from an XML file

Use XML as an intermediate format for managing your UltraLite database.

Prerequisites

UltraLite cannot use an arbitrary XML file. The %SQLANY16%\Bin32 and %SQLANY16%\Bin64 directories contains a usm.xsd file, containing the schema definition. Use this file to review the XML format.

Context and remarks

You can use XML to:

- Load data into a new database with a different set of database properties/options.
- Upgrade the schema from a database created by a previous version of UltraLite.
- Create a text version of your UltraLite database.

Task

1. Save the XML file to a directory of your choosing. You can either:

   - Export/unload a database to an XML file. If you are unloading a SQL Anywhere database, use any of the supported export methods.
   - Take XML output from another source—that source could be another relational database or even a web site where transactions are recorded to a file. You must always ensure that the format of the XML meets the UltraLite requirements.
2. Run the ulload utility, including any necessary options.

**Results**

The database is created.

**Next**

You can now connect to the database and build tables and other database objects.

**Example**

To create a new UltraLite database in the file `sample.udb` from the table formats and data in `sample.xml`, run the following command:

```
ulload -c DBF=sample.udb sample.xml
```

**See also**

- “Relational data exported as XML” [SQL Anywhere Server - SQL Usage]
- “Creating an UltraLite database with the Create Database Wizard” on page 19
- “UltraLite database creation using a command prompt” on page 20
- “Conversion from a SQL Anywhere database to an UltraLite database” on page 29
- “UltraLite upgrades” [SQL Anywhere 16 - Changes and Upgrading]
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “UltraLite database creation parameters” on page 23

**UltraLite database creation on a first connection**

You can program your application to create a new UltraLite database if one cannot be detected at connection time. The application can then use SQL to create tables, indexes, foreign keys, and so on. To populate the database, synchronize with a consolidated database.

**Considerations**

By adding the additional database creation and SQL code, your application size can grow considerably. However, this option can simplify deployment because you only need to deploy the application to the device. In some pre-production development cycles, you may want to delete and reconstruct the database on your device for testing purposes.

**See also**

- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “UltraLite and UltraLite Java edition database creation and connection approaches” [UltraLite - Java Programming]
UltraLite database creation parameters

Creation parameters are used to configure the UltraLite database when you first create it. You can only change these settings by recreating the database.

There are several options you can control when creating your UltraLite database. These are designed to help with the wide variety of UltraLite uses. Most parameters specified at creation time cannot be changed later.

You can specify creation parameters when creating a database using the ulinit or ulload utility, and from the supported client interfaces.

Boolean creation parameters are turned on with YES, Y, ON, TRUE, T, or 1, and are turned off with any of NO, N, OFF, FALSE, F, and 0. The parameters are case insensitive.

UltraLite creation parameters are specified in a semicolon separated string when creating a database from a programming interfaces and as a separate command line parameters when using a command line utility. For example:

```
ulinit --case --utf8_encoding=1 test.udb
```

Alternatively, you can specify multiple -c options.

You can further configure other aspects of your database with database options and connection parameters.

Accessing creation parameter values

You cannot change creation parameter values after you have created a database. However, you can view the corresponding database properties in Sybase Central.

For the UltraLiteJ API, you can use the getCreationString method to view the creation string registered with the setCreationString method.

For other UltraLite APIs, you can access the database properties programmatically from your UltraLite application by calling the GetDatabaseProperty function appropriate to the UltraLite API.

See also

- “UltraLite database options” on page 181
- “UltraLite database connections” on page 31
- “UltraLite connection parameters” on page 156
- “Reading database properties” on page 35
- “ULConnection.GetDatabaseProperty method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ConfigPersistent.getCreationString method [Android] [UltraLiteJ]” [UltraLite - Java Programming]
- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” [UltraLite - Java Programming]
UltraLite character sets

The results of comparisons on strings, and the sort order of strings, in part depends on the character set, collation, and encoding properties of the database.

Choosing the correct character set, collation, and encoding properties for your database is primarily determined by:

- The desired sort order. Choose the collation that best sorts the characters you intend to store in your database.
- The platform of your device. Requirements among supported devices can vary, and some require that you use UTF-8 to encode your characters. If you need to support multiple devices, you need to determine whether a database can be shared.
- If you are synchronizing data, which languages and character sets are supported by the consolidated database. You must ensure that the character sets used in the UltraLite database and the consolidated database are compatible. Otherwise, data could be lost or become altered in unexpected ways if characters in one database's character set do not exist in the other's character set. If you have deployed UltraLite in a multilingual environment, you should also use UTF-8 to encode your UltraLite database.

When you synchronize, the MobiLink server converts characters as follows:

1. The UltraLite database characters are converted to Unicode.
2. The Unicode characters are converted into the consolidated database's character set.

Character sets and collations for UltraLite Java edition

UltraLite Java edition databases are encoded as UTF-8 (Unicode). The collation is the default sort order for Java and is equivalent to the UTF8BIN collation supported by SQL Anywhere. During synchronization with a MobiLink server, UltraLite notifies MobiLink that it uses the UTF8 character set and collation.

See also

- “UltraLite platform requirements for character set encoding” on page 24
- “UltraLite collation creation parameter” on page 135
- “UltraLite utf8_encoding creation parameter” on page 155
- “Character sets” [SQL Anywhere Server - Database Administration]
- “UltraLite connection parameters” on page 156
- “Character set considerations” [MobiLink - Server Administration]
- “UltraLite case creation parameter” on page 133
- “Database security” on page 26

UltraLite platform requirements for character set encoding

Each platform has specific character set and encoding requirements.
**Windows desktop and Windows Mobile**

When using a UTF-8 encoded database on Windows, you should pass wide characters to the database. If you use UTF-8 encoding on these platforms, UltraLite expects that non-wide string parameters are UTF-8 encoded, which is not a natural character set to use on Windows. The exception is for connection strings, where string parameters are expected to be in the active code page. However, by using wide characters, you can avoid this complication.

See also

- “UltraLite utf8_encoding creation parameter” on page 155
- “Character sets” [SQL Anywhere Server - Database Administration]
- “UltraLite connection parameters” on page 156
- “Character set considerations” [MobiLink - Server Administration]
- “Database security” on page 26

**UltraLite supported collations**

The following table lists the supported CHAR collations in UltraLite. You can also generate the list by executing the following command:

```
ulinit -Z
```

UltraLite Java edition databases only supports the UTF8BIN collation.

<table>
<thead>
<tr>
<th>Collation label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1250LATIN2</td>
<td>Code Page 1250, Windows Latin 2, Central/Eastern European</td>
</tr>
<tr>
<td>1250POL</td>
<td>Code Page 1250, Windows Latin 2, Polish</td>
</tr>
<tr>
<td>1251CYR</td>
<td>Code Page 1251, Windows Cyrillic</td>
</tr>
<tr>
<td>1252LATIN1</td>
<td>Code Page 1252, Windows Latin 1, Western</td>
</tr>
<tr>
<td>1252NOR</td>
<td>Code Page 1252, Windows Latin 1, Norwegian</td>
</tr>
<tr>
<td>1252SPA</td>
<td>Code Page 1252, Windows Latin 1, Spanish</td>
</tr>
<tr>
<td>1252SWEFIN</td>
<td>Code Page 1252, Windows Latin 1, Swedish/Finnish</td>
</tr>
<tr>
<td>1253ELL</td>
<td>Code Page 1253, Windows Greek, ISO8859-7 with extensions</td>
</tr>
<tr>
<td>1254TRK</td>
<td>Code Page 1254, Windows Turkish, ISO8859-9 with extensions</td>
</tr>
<tr>
<td>1254TRKALT</td>
<td>Code Page 1254, Windows Turkish, ISO8859-9 with extensions, I-dot e als I-no-dot</td>
</tr>
<tr>
<td>1255HEB</td>
<td>Code Page 1255, Windows Hebrew, ISO8859-8 with extensions</td>
</tr>
</tbody>
</table>
Database security

This topic has been updated for build 1823.

You have the option to encrypt or obfuscate your databases. Encryption provides secure representation of the data in the database whereas obfuscation only prevents casual observation of the contents of the database.

By default, databases are not encrypted or obfuscated. Text and binary columns can be read when using a viewing tool such as a hex editor. Consider the following options if you do not want your data stored as plain text:

<table>
<thead>
<tr>
<th>Collation label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1256ARA</td>
<td>Code Page 1256, Windows Arabic, ISO8859-6 with extensions</td>
</tr>
<tr>
<td>1257LIT</td>
<td>Code Page 1257, Windows Baltic Rim, Lithuanian</td>
</tr>
<tr>
<td>874THAIBIN</td>
<td>Code Page 874, Windows Thai, ISO8859-11, binary ordering</td>
</tr>
<tr>
<td>932JPN</td>
<td>Code Page 932, Japanese Shift-JIS with Microsoft extensions</td>
</tr>
<tr>
<td>936ZHO</td>
<td>Code Page 936, Simplified Chinese, PRC GBK</td>
</tr>
<tr>
<td>950ZHO_HK</td>
<td>Code Page 950, Traditional Chinese, Big 5 Encoding with HKSCS</td>
</tr>
<tr>
<td>950ZHO_TW</td>
<td>Code Page 950, Traditional Chinese, Big 5 Encoding</td>
</tr>
<tr>
<td>EUC_CHINA</td>
<td>Simplified Chinese, GB 2312-80 Encoding</td>
</tr>
<tr>
<td>EUC_JAPAN</td>
<td>Japanese EUC JIS X 0208-1990 and JIS X 0212-1990 Encoding</td>
</tr>
<tr>
<td>EUC_TAIWAN</td>
<td>Code Page 964, EUC-TW Encoding</td>
</tr>
<tr>
<td>ISO1LATIN1</td>
<td>ISO8859-1, ISO Latin 1, Western, Latin 1 Ordering</td>
</tr>
<tr>
<td>ISO9LATIN1</td>
<td>ISO8859-15, ISO Latin 9, Western, Latin 1 Ordering</td>
</tr>
<tr>
<td>ISO_1</td>
<td>ISO8859-1, ISO Latin 1, Western</td>
</tr>
<tr>
<td>ISO_BINENG</td>
<td>Binary ordering, English ISO/ASCII 7-bit letter case mappings</td>
</tr>
<tr>
<td>UTF8BIN</td>
<td>UTF-8, 8-bit multibyte encoding for Unicode, binary ordering</td>
</tr>
</tbody>
</table>
- **Obfuscation**  This option provides protection against casual attempts to access data in the database but does not provide as much security as strong encryption. Obfuscation has a minimal performance impact. You do not need any special configuration to use simple obfuscation on your device.

- **AES 256-bit encryption**  This option encrypts databases with an AES 256-bit algorithm. Strong encryption provides security against skilled and determined attempts to gain access to the data but has a significant performance impact. You do not need any special configuration to use AES encryption on your device.

- **AES 256-bit FIPS 140-2 certified encryption (Windows only)**  Encryption libraries certified with the FIPS 140-2 US and Canadian government standard are provided. AES FIPS-certified encryption requires that you configure your device appropriately.

  FIPS-certified encryption requires a separate license. All strong encryption technologies are subject to export regulations.

Consider the effects of database cache size when choosing to encrypt or obfuscate databases. There is an overhead increase between 5-10%, which can result in decreased performance. The exact effect on performance depends on the size of your cache. If your cache is too small, encryption can add significant overhead. However, if your cache is sufficiently large, you may not see any difference at all. For more information about cache sizes, see “Cache size adjustment for an UltraLite database” on page 439.

### Database obfuscation

To obfuscate data, specify `obfuscate=1` as a database creation parameter when you create your database. End users do not need to supply a corresponding connection parameter.

To obfuscate data with the UltraLiteJ API, use the `ConfigPersistent.enableObfuscation` method during database creation.

### Database encryption

Encryption keys should contain a combination of characters, numbers, and special symbols to be effective. Long encryption keys reduce the chances of someone guessing the key.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>After the database is encrypted, the encryption key cannot be recovered.</td>
</tr>
</tbody>
</table>

Using Sybase Central wizards, you can specify UltraLite database encryption options during creation by clicking the **Encrypt the Database** option and then clicking **Use Strong Encryption and AES**.

Using the ulinit utility, you can specify the encryption key with the DBKEY parameter. Use the FIPS creation parameter to specify whether to use FIPS-certified encryption.

UltraLite API encryption options are available when creating a database. For UltraLite Java edition databases, specify the database key using the `enableAesDBEncryption` and `setEncryptionKey` methods instead of the DBKEY parameter.
Caution

You can change the encryption key after the database has been created but only under extreme caution. This operation is costly and is non-recoverable. You can lose your database entirely if your operation terminates mid-course.

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.

For more information, see:

- “ULConnection.ChangeEncryptionKey method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULConnection.ChangeEncryptionKey method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Connection.changeEncryptionKey method [UltraLiteJ]” [UltraLite - Java Programming]

The DBKEY parameter must be supplied when connecting to the database; otherwise, the connections fail. Encryption keys should be treated as sensitive information.

See also

- “UltraLite obfuscate creation parameter” on page 143
- “UltraLite DBKEY connection parameter” on page 166
- “UltraLite fips creation parameter” on page 140
- “UltraLite database creation approaches” on page 19
- “UltraLite database creation parameters” on page 23
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “ConfigPersistent.enableObfuscation method [UltraLiteJ]” [UltraLite - Java Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “ConfigPersistent.setCreationString method [Android] [UltraLiteJ]” [UltraLite - Java Programming]
- “ConfigPersistent.enableAesDBEncryption method [UltraLiteJ]” [UltraLite - Java Programming]
- “ConfigPersistent.getEncryptionKey method [UltraLiteJ]” [UltraLite - Java Programming]
- “ConfigPersistent.setEncryptionKey method [UltraLiteJ]” [UltraLite - Java Programming]
- “Separately licensed components” [SQL Anywhere 16 - Introduction]
Conversion from a SQL Anywhere database to an UltraLite database

Create an UltraLite database from a SQL Anywhere reference database by running the ulinit utility with the -a option. The new database is created with the same settings as those in the reference database where possible.

The SQL Anywhere reference database acts as a database template, and uses the following settings to create an UltraLite database schema:

- Database configuration, such as the collation sequence
- Table definitions
- Synchronization publications

You can include data, and choose the columns, tables, and indexes as part of a publication in the reference database.

**Note**
To initialize an UltraLite database from an RDBMS other than SQL Anywhere, use the Create Synchronization Model Wizard in Sybase Central, and connect to a consolidated database when prompted to obtain the schema information.

Conversion considerations

Prior to running the ulinit utility, consider if the following reference database tasks are required:

- **Add tables, keys, indexes, and synchronization publications as needed**  
  Add the tables and set primary keys as needed. You can also assign foreign key relationships that you need within your UltraLite application.

  Indexes can improve performance dramatically, particularly on slow devices. Primary key columns are automatically indexed, but other types of columns are not.

**Tip**
If your UltraLite application frequently retrieves information in a particular order, consider adding an index to your reference database specifically for this purpose.

Use synchronization publications to synchronize different tables at different times. You can use multiple synchronization publications to define table subsets and set the synchronization priority with them.

- **Update database options or table schema that may have undesired effects**  
  For example, if a column in the SQL Anywhere database includes a clause that UltraLite does not support, the default value is ignored and the UltraLite default is specified for the new database.

- **Change the collation sequence if it is not supported by UltraLite**  
  UltraLite uses the name of the collation sequence that is defined in the reference database. You can still choose to use UTF-8 to encode the database by setting the utf8_encoding property.
To see a list of collations and corresponding codepages supported by UltraLite, run ulinit with the -Z option at a command prompt. If the reference database uses a collation sequence that is not supported, such as UCA for CHAR collation sequences, change the collation sequence to one that is supported by performing the following steps:

1. Use the Unload utility to unload the SQL Anywhere reference database.

2. Create a new SQL Anywhere database with a different collation and run the reload.sql script through Interactive SQL.

See also

- “Unload utility (dbunload)” [SQL Anywhere Server - Database Administration]
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite tables and columns” on page 46
- “When to use an index” on page 54
- “Publishing data in UltraLite” on page 72
- “Index scan creation and maintenance” on page 441
- “UltraLite database creation approaches” on page 19

Example

The following command creates a new UltraLite database named customer.udb from an existing SQL Anywhere reference database defined in the MySADb data source. Tables in the reference database are defined in TestPublication. The created UltraLite database contains all the same database options and tables contained in TestPublication, and is encrypted with the mykey encryption key.

```bash
ulinit -a "DSN=MySADb;UID=JimmyB;PWD=secret" -n TestPublication -k mykey customer.udb
```

The following command creates a new UltraLite database named customer.udb from an existing SQL Anywhere database named MySource.db. The tables and indexes in the created database match those contained in the Pub1 schema publication. The Pub2 synchronization publication is created in the UltraLite database.

```bash
ulinit -a DBF=MySource.db;UID=JimmyB;PWD=secret customer.udb -n Pub1 -s Pub2
```
UltraLite database connections

Applications that use a database must establish a connection to that database before transactions can occur. An application can be an UltraLite utility, a connection window, or your own custom application.

By connecting to an UltraLite database, you form a channel through which all activity from the application takes place. Each connection attempt creates a database specific SQL transaction.

See also

- “UltraLite connection parameters” on page 156
- UltraLite C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- UltraLiteJ: “UltraLite and UltraLite Java edition database creation and connection approaches” [UltraLite - Java Programming]

UltraLite connection strings and parameters

A connection string is a set of connection parameters that is passed from an application so that a database connection can be defined and established. Some parameters are always required to open a connection while others are used to adjust database features for a single connection.

Connection strings are defined as keyword=value pairs in a semicolon delimited list. The following example illustrates a connection string fragment that specifies a database file name, user ID, and password:

```
DBF=myULdb.udb;UID=J Doe;PWD=token
```

Methods of supplying these parameters to a database can vary depending on whether you are connecting from an UltraLite utility or an UltraLite application.

UltraLite command line utilities typically use a connection string if a connection to a database is required.

UltraLite applications can be developed using an UltraLite API to read connection parameter values from a stored file or in the application code. You can supply fixed connection strings when user authentication is not required, or prompt users to supply parameter values at connection time.

When a connection string has been assembled, it is passed to the UltraLite runtime for processing. The connection to the database is granted when the connection attempt is validated. Connection failures can occur if the database file does not exist, or the authentication was not successful.

UltraLite generates an error when it encounters an unrecognized connection parameter.

Precedence of connection parameters for UltraLite administration tools

UltraLite administration tools follow a specific order of connection parameter precedence:
• OS-specific options take precedence over nonspecific options. For example: CE_DBF takes precedence over DBF on Windows CE devices.

• If specified, the CE_FILE, desktop, device, and NT_FILE parameters always take precedence over DBF.

• If you supply duplicate parameters in a connection string, the last one supplied is used. All others are ignored.

• Parameters in the connection string take precedence over those supplied in the ULSQLCONNECT environment variable or a connection object.

• If no value is supplied for both UID and PWD in either the connection string or ULSQLCONNECT, the defaults of UID=DBA and PWD=sql are assumed.

Limitations
Any leading and/or trailing spaces in connection string parameter values are ignored. Connection parameter values cannot include leading single quotes ('), leading double quotes ("), or semicolons (;).

See also
- “UltraLite connection parameters” on page 156
- UltraLite C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- UltraLiteJ: “UltraLite and UltraLite Java edition database creation and connection approaches” [UltraLite - Java Programming]

Example
You can type a connection string on a single line with the parameter names and values separated by semicolons:

    parameter1=value1;parameter2=value2

The UltraLite runtime ensures that the parameters are assembled into a connection string before establishing a connection with it. For example, if you use the ulload utility, the following connection string is used to load new XML data into an existing database. You cannot connect to the database file until you supply this string:

    ulload -c "DBF=sample.udb;UID=DBA;PWD=sql" sample.xml

UltraLite connection parameters and the ULSQLCONNECT environment variable
Use the ULSQLCONNECT environment variable to avoid having to supply the same connection parameters repeatedly to Interactive SQL during development. Interactive SQL is the only tool that supports the ULSQLCONNECT environment variable; you cannot use the variable for custom applications.
The ULSQLCONNECT environment variable is optional, and is not specified by default. You can set the ULSQLCONNECT environment variable to contain a list of parameters defined as keyword=value pairs in a semicolon delimited list.

The supplied values become defaults for the UltraLite desktop administration tools. If Interactive SQL requires any additional parameters or if you need to override default values set with this environment variable, ensure you set these values. User supplied values always take precedence over this environment variable.

**Caution**

Do not use the pound character (#) as an alternative to the equal sign; the pound character is ignored by dbisql. All platforms supported by UltraLite allow you to use = inside an environment variable setting.

**See also**

- “Precedence of connection parameters for UltraLite administration tools” on page 31

**Example**

In this example, you use the ULSQLCONNECT environment variable to connect to a file named c:\database\myfile.udb and authenticate the user demo with the password test.

Set the following variable in your ULSQLCONNECT environment variable:

```
set ULSQLCONNECT="DBF=c:\database\myfile.udb;UID=demo;PWD=test"
```

By setting this environment variable, you no longer need to use the dbisql -c connection option for these defaults values—unless you need to override these values.

---

### UltraLite file path formats in connection parameters

The physical storage of your device determines whether the database is saved as a file and what naming conventions you must follow when identifying your database.

**Note**

Use absolute file paths when using the UltraLite engine to support multi-process access to a database since the engine may be started in different locations.

The DBF parameter is most appropriate when targeting a single deployment platform or when using UltraLite desktop administration tools. For example:

```
ulload -c DBF=sample.udb sample.xml
```

**Tip**

You can use the UltraLite administration tools to administer databases already deployed to an attached device. See “Windows Mobile” on page 34.

Otherwise, if you are writing a cross-platform application, use the platform-specific (CE_DBF or NT_DBF) file connection parameters to construct a universal connection string. For example:
Connection = DatabaseManager::OpenConnection("UID=JDoe;PWD=ULdb;CE_DBF=\database\MyCEDB.udb;NT_FILE=MyDB.udb")

Desktop

Desktops allow either absolute or relative paths.

Windows Mobile

Windows Mobile devices require absolute paths.

You can administer a Windows Mobile database on either the desktop or the attached device. To administer a database on a Windows Mobile device, ensure you prefix the absolute path with wce:\. For example, using the unload utility:

```
unload -c DBF=wce:\UltraLite\myULdb.udb c:\out\ce.xml
```

In this example, UltraLite unloads the database from the Windows Mobile device to the ce.xml file in the Windows desktop folder of c:\out.

If you are using the unloadold or unload utilities to administer a database on the Windows Mobile device directly, UltraLite cannot back up the database before the unload or action occurs. You must perform this action manually before running these utilities.

See also

- “UltraLite DBF connection parameter” on page 165
- “UltraLite NT_FILE connection parameter” on page 171
- “UltraLite CE_FILE connection parameter” on page 161
UltraLite database tasks and features

Reading database properties

You can change the settings of any database property that does not correspond to a database creation parameter.

Prerequisites

There are no prerequisites for this task.

Context and remarks

Each UltraLite API contains a GetDatabaseProperty method that you can use in your applications to access database properties.

You can also access database properties by calling the DB_PROPERTY SQL function.

Task

1. Using Sybase Central, connect to the database.
2. Right-click the database and click Properties.

Results

In the Database Properties window, database properties are listed on the General and Synchronization Information tabs. On the Synchronization Information tab, the database properties are listed alphabetically by the property name. To sort database properties by the value, click the Value column.

See also

- “UltraLite database properties” on page 177
- “ULConnection.GetDatabaseProperty method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULConnection.GetDatabasePropertyInt method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
- “DB_PROPERTY function [System]” on page 328

Accessing database options

View and change database options to configure database behavior.
Prerequisites

There are no prerequisites for this task.

Context and remarks

You can view and change the setting of persistent database options from Sybase Central. Temporary UltraLite database options cannot be viewed or set from Sybase Central.

Database options can be set or modified at any time. Temporary database options only persist while the database is running.

Option values are set by using the SET OPTION SQL statement.

Task

1. Using Sybase Central, connect to the database.
2. Right-click the database and click Options.
3. To set or reset an option, type a new value in the Value field.
4. Click Set Now or Reset Now to commit the change.

Results

The database option setting is changed and saved.

See also

- “UltraLite database options” on page 181
- “SET OPTION statement [UltraLite]” on page 431
- “ULConnection.SetDatabaseOption method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULConnection.SetDatabaseOptionInt method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseSchema.SetDatabaseOption method [UltraLite.NET]” [UltraLite -.NET Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Reading database properties” on page 35

UltraLite event notifications

UltraLite now supports events and notifications. A notification is a message that is sent when an event occurs, also providing additional parameter information. UltraLite has system events and events can also be user-defined.

Event notifications allow you to provide coordination and signaling between connections or applications connected to the same database. Notifications are managed in queues: either a connection's default queue or, optionally, queues that are explicitly created and named. When an event occurs, notifications are sent to registered queues (or connections).
Each connection manages its own notification queues. Named queues can be created for any connection.

Using predefined system events this feature also provides "triggers" for changes to data, such as when a change is made to a table, for example, or signaling when a synchronization has occurred. Predefined events include:

- Commit
- SyncComplete
- TableModified

User events may also be defined and triggered by an application.

APIs for events and notifications are provided in each supported language. Additionally, a SQL function is provided to access the API functionality.

**Events**

<table>
<thead>
<tr>
<th>Event</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>Signaled upon completion of a commit.</td>
</tr>
<tr>
<td>SyncComplete</td>
<td>Signaled upon completion of a sync.</td>
</tr>
<tr>
<td>TableModified</td>
<td>Triggered when rows in a table are inserted, updated, or deleted. One event is signaled per request, no matter how many rows were affected by the request when registering for the event. The object_name parameter specifies the table to monitor. A value of &quot;*&quot; means all tables in the database. The table_name notification parameter is the name of the modified table.</td>
</tr>
</tbody>
</table>

```c
note_info.event_name = "SyncComplete";
note_info.event_name_len = 12;
note_info.parms_type = ul_ev_note_info::P_NONE;

note_info.event_name = "TableModified";
note_info.event_name_len = 13;
note_info.parms_type = ul_ev_note_info::P_TABLE_NAME;
note_info.parms = table->name->data;
note_info.parms_len = table->name->len;
```

**Working with queues**

Queues can be created and destroyed.

CreateNotificationQueue creates an event notification queue for the current connection. Queue names are scoped per-connection, so different connections can create queues with the same name. When an event notification is sent, all queues in the database with a matching name receive a separate instance of the notification. Names are case insensitive. A default queue is created on demand for each connection if no queue is specified. This call fails with an error if the name already exists for the connection or isn't valid.
DestroyNotificationQueue destroys the given event notification queue. A warning is signaled if unread
notifications remain in the queue. Unread notifications are discarded. A connection's default event queue,
if created, is destroyed when the connection is closed.

Working with events

DeclareEvent declares an event which can then be registered for and triggered. UltraLite predefines some
system events triggered by operations on the database or the environment. The event name must be
unique and names are case insensitive. Returns true if the event was declared successfully, false if the
name is already used or invalid.

RegisterForEvent registers a queue to receive notifications of an event. If no queue name is supplied, the
default connection queue is implied, and created if required. Certain system events allow specification of
an object name to which the event applies. For example, the TableModified event can specify the table
name. Unlike SendNotification, only the specific queue registered will receive notifications of the event;
other queues with the same name on different connections will not (unless they are also explicitly
registered). Returns true if the registration succeeded, false if the queue or event does not exist.

TriggerEvent triggers an event and sends a notification to all registered queues. Returns the number of
event notifications sent. Parameters may be supplied as name=value; pairs.

Working with notifications

SendNotification sends a notification to all queues in the database matching the given name (including
any such queue on the current connection). This call does not block. Use the special queue name "*" to
send to all queues. Returns the number of notifications sent (the number of matching queues). Parameters
may be supplied as name=value; pairs.

GetNotification reads an event notification. This call blocks until a notification is received or until the
given wait period expires. To cancel a wait, send another notification to the given queue or use
CancelGetNotification. After reading a notification, use ReadNotificationParameter to retrieve additional
parameters. Returns true if an event was read, false if the wait period expired or was canceled.

GetNotificationParameter gets a named parameter for the event notification just read by
GetNotification. Only the parameters from the most-recently read notification on the given queue are
available. Returns true if the parameter was found, false if the parameter was not found.

CancelGetNotification cancels any pending GetNotification calls on all queues matching the given
name. Returns the number of affected queues (not necessarily the number of blocked reads).

Other considerations

- Notification queue and event names are limited to 32 characters.
- To govern system resources, the number of notifications is limited. When this limit is exceeded,
  SQLE_EVENT_NOTIFICATION_QUEUE_FULL is signaled and the pending notification is
discarded.
Isolation levels

Isolation levels define the degree to which the operations in one transaction are visible to the operations in other concurrent transactions. UltraLite uses the default isolation level, read_committed, for connections. The default UltraLite isolation level aids data consistency by isolating uncommitted rows.

<table>
<thead>
<tr>
<th>This isolation level...</th>
<th>Has these characteristics...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0—read_uncommitted</td>
<td>● Allow dirty reads, non-repeatable reads, and phantom rows.</td>
</tr>
<tr>
<td></td>
<td>● No guarantee that concurrent transactions will not modify row or roll back changes to rows</td>
</tr>
<tr>
<td>1—read_committed</td>
<td>● Allow non-repeatable reads and phantom rows</td>
</tr>
<tr>
<td></td>
<td>● Prevent dirty reads</td>
</tr>
<tr>
<td></td>
<td>● No guarantee that query results will not change during transactions</td>
</tr>
</tbody>
</table>

You can change the isolation level from read_committed to read_uncommitted using one of following approaches:

- Use the SET OPTION SQL statement and the isolation_level database option.
  For example, the following statement sets the isolation level to read uncommitted:

  ```sql
  SET OPTION isolation_level = 'READ_UNCOMMITTED'
  ```

- For the UltraLite C++ API, use the ULConnection.SetDatabaseOption method to change the isolation level.
  For the UltraLite.NET API, use the ULConnection.BeginTransaction or ULDatabaseSchema.SetDatabaseOption methods to create a transaction with the read_committed isolation level.
  For the UltraLiteJ API, use the Connection.setOption method.

  **Note**
  UltraLite Java edition databases only support the read_uncommitted isolation level. See “Data synchronization on a BlackBerry smartphone” [UltraLite - Java Programming].

Concurrency and locking for UltraLite Java edition databases

- **Locking** Two different connections cannot modify the same row concurrently. If two connections attempt to operate on the same row, the second connection receives an error and is prohibited from modifying the row until the first connection commits or rolls back its current transaction.

- **Visibility** One connection's operation on the database is immediately visible to other connections.
Characteristics of the read_uncommitted isolation level

The following side effects are possible when UltraLite operates at an isolation_level of 0 (read_uncommitted):

- Applications can read uncommitted data (dirty reads). In this scenario, transactions may access rows in the database that are not committed and may still get rolled back by another transaction. This phenomena can result in phantom rows (rows that get added after the original query, making the result set returned in a repeated, duplicate query different).
  
  For a tutorial that demonstrates the effects of dirty reads, see “Tutorial: Understanding dirty reads” [SQL Anywhere Server - SQL Usage].
  
  For a tutorial that demonstrates a phantom row, see “Tutorial: Understanding phantom rows” [SQL Anywhere Server - SQL Usage].

- Applications can perform non-repeatable reads. In this scenario, an application reads a row from the database, and then goes on to perform other operations. Then a second application updates/deletes the row and commits the change. If the first application attempts to re-read the original row, it receives either the updated information or discovers that the original row was deleted.
  
  For a tutorial that demonstrates the effects of non-repeatable reads, see “Tutorial: Understanding non-repeatable reads” [SQL Anywhere Server - SQL Usage].
Note
Reading or fetching a row does not lock the row. If connection A fetches but does not modify a row, connection B can still modify the row.

2. As A modifies the current row, it changes the copy in the buffer. The copy in the buffer is written back into the database when connection A calls an Update method or closes the result set.

3. A write lock is placed on the row to prevent other transactions from modifying it. This modification is uncommitted, until connection A performs a commit.

4. Depending on the modification, if connection B fetches the current row, it may experience the following:

<table>
<thead>
<tr>
<th>Connection A's modification</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row has been deleted.</td>
<td>Connection B gets the next row in the result set.</td>
</tr>
<tr>
<td>Row has been modified.</td>
<td>Connection B gets the latest copy of the row.</td>
</tr>
</tbody>
</table>

1 Queries used by Connection A and B do not contain temporary tables. Temporary tables can cause other side effects.

Validating an UltraLite database

You should periodically check that your database is valid by using tools such as the Validate Database Wizard in Sybase Central, the UltraLite Validate Database utility, or the ValidateDatabase method in your UltraLite API.

Prerequisites

There are no prerequisites for this task.

Context and remarks

Database file corruption may not be reported until the database server tries to access the affected part of the database.

Caution
Database validation should be performed while no connections are making changes to the database; otherwise, errors indicating database corruption might be reported even though no corruption actually exists.
You can validate an UltraLite database using any of the following methods:

- The **Validate Database Wizard** in Sybase Central.
- The `ulvalid` command line utility.
- The `ValidateDatabase` method in your UltraLite API.

UltraLite Java edition database validation is not supported.

**Task**

1. In the left pane of Sybase Central, click the UltraLite database.
2. Click **File » Validate Database**.
3. Follow the instructions in the **Validate Database Wizard**.

**Results**

The database is validated.

**See also**

- “ULConnection.ValidateDatabase method [UltraLite C++]” in [UltraLite - C and C++ Programming]
- “ULConnection.ValidateDatabase method [UltraLite.NET]” in [UltraLite - .NET Programming]
- “Connection.validateDatabase method [Android] [UltraLiteJ]” in [UltraLite - Java Programming]
- “UltraLite Validate Database utility (ulvalid)” on page 219

**UltraLite and UltraLite Java edition database back up and recovery**

If an application using an UltraLite or UltraLite Java edition database stops unexpectedly, the database automatically recovers to a consistent state when the application is restarted. All committed transactions flushed to memory before the unexpected failure are present in the database. All transactions not flushed at the time of the failure are rolled back.

An UltraLite database does not use a transaction log to perform recovery. Instead, UltraLite stores state information for every row to determine the fate of a row when recovering.

**Backups**

UltraLite provides protection against system failures, but not from media failures. The best way to make a backup of an UltraLite application is to synchronize with a consolidated database. To restore an UltraLite or UltraLite Java edition database, start with an empty database and populate it from the consolidated database through synchronization.
See also

- “UltraLite database row state management” on page 453
- “Flush single or grouped transactions” on page 455
- “UltraLite as a MobiLink client” on page 63
UltraLite database schemas

The logical framework of the database is known as a schema.

UltraLite database schemas

You can upgrade the schema of an UltraLite database with the appropriate Data Definition Language (DDL) statements or by using the ALTER DATABASE SCHEMA FROM FILE statement to modify the schema definition using a SQL script.

Schema changes can take a considerable amount of time. For example, all rows in the associated table must be updated when the column type is changed. DDL statements successfully execute when there are not any:

- Uncommitted transactions.
- Other active uses of the database, such as synchronization, prepared but unreleased statements, or executing database operations.

When the DDL statement is executing, any other attempt to use the database is blocked until the DDL statement completes the schema change.

UltraLite Java edition database schemas

The UltraLite Java edition database schema is maintained as a catalog of system tables that hold the metadata for the UltraLite Java edition database. System table metadata include:

- **Table definitions** Stored in the systable system table.
- **Column definitions** Stored in the syscolumn system table.
- **Index definitions** Stored in the sysindex and sysindexcolumn system tables.
- **Publication definitions** Stored in the syspublications and sysarticles system tables.
- **Foreign key definitions** Stored in the sysforeignkey and sysfkcol system tables.
- **User names and passwords** Stored in the sysuldata system tables.
UltraLite tables and columns

Tables are used to store data and define the relationships for data in them. Tables consist of rows and columns. Each column carries a particular kind of information, such as a phone number or a name, while each row specifies a particular entry.

When you first create an UltraLite database, the only tables you will see are the system tables. System tables hold the UltraLite schema. You can hide or show these tables from Sybase Central as needed.

You can then add new tables as required by your application. You can also browse data in those tables, and copy and paste data among existing tables in the source database or even among other open destination databases.

In UltraLite, you can only create base tables, which you declare to hold persistent data. UltraLite does not support global temporary or declared temporary tables.

Row packing and table definitions

UltraLite works with rows in two formats:

- **Unpacked rows** are the uncompressed format. Each row must be unpacked before individual column values can be read or written.

- **Packed rows** are the compressed representation of the unpacked row, where each of the column values is compressed so that the entire row takes up as little memory as possible. The size of a packed row depends entirely on the values in each column: for example, two rows can belong to the same table, but can differ significantly in their packed size. Note also that LONG BINARY and LONG VARCHAR columns are stored separate from the packed row.

UltraLite has a limitation that a packed row must fit on a database page. Since LONG BINARY and LONG VARCHAR columns are not stored with the packed row, they can exceed the page size.

It is important to understand that table definitions describe the row before the UltraLite runtime packs the data. Because the size of a packed row depends on the values in each column, you cannot readily pre-determine from the table definition whether the packed row requirement is satisfied. For this reason,
UltraLite allows you to define a table where an unpacked row would not fit on a page. To know if a row fits on a page, you must try inserting or updating the row itself; if a row does not fit, UltraLite detects and reports this error.

**Note**
You cannot declare tables to be any large size. UltraLite maintains a declared table row size limit of 64 KB. If you try to define a table where an unpacked row can exceed this maximum, UltraLite generates a SQL error code of SQLE_MAX_ROW_SIZE_EXCEEDED (-1132).

See also
- “UltraLite page_size creation parameter” on page 144
- “Database tables” [SQL Anywhere 16 - Introduction]
- “Database creation” [SQL Anywhere Server - Database Administration]
- “UltraLite system tables” on page 220

Creating UltraLite tables
You can create base tables to hold your persistent relational data.

**Prerequisites**
Tables in UltraLite applications must include a primary key. Primary keys are also required during MobiLink synchronization, to associate rows in the UltraLite database with rows in the consolidated database.

**Task**
1. Using Sybase Central, connect to the UltraLite database.
2. In the left pane, right-click Tables and click New » Table.
3. In the What Do You Want To Name The New Table field, type the new table name.
4. Click Finish.
5. From the File menu, click Save.

**Results**
The table is created. The table and any data it contains exist until you explicitly delete the data or drop the table.

**Next**
Add columns or create indexes.
Adding a column to an UltraLite table

Columns can be added to an UltraLite table after it has been created.

Prerequisites

If the table already holds data, you can only add a column if the column definition includes a default value or allows NULL values.

Task

1. Using Sybase Central, connect to the UltraLite database.
2. In the left pane, double-click Tables.
3. Double-click a table.
4. Click the Columns tab, right-click the white space below the table and click New » Column.
5. Set the attributes for the new column and then save your changes.

Results

The column is added to the table.

See also

- “Object name considerations” [SQL Anywhere Server - Database Administration]
- “UltraLite SQL data types” on page 275
- “Column data type considerations” [SQL Anywhere Server - Database Administration]
- “CREATE TABLE statement [UltraLite]” on page 411
- “ALTER TABLE statement [UltraLite]” on page 401

Altering UltraLite column definitions

Change the structure of column definitions for a table by altering various column attributes or deleting columns entirely.

Prerequisites

The modified column definition must suit the requirements of any data already stored in the column. For example, you cannot alter a column to disallow NULL if the column already has a NULL entry.
Task

1. Using Sybase Central, connect to the UltraLite database.
2. In the left pane, double-click Tables.
3. Double-click a table.
4. Click the Columns tab and alter the column attributes.
5. From the File menu, click Save Table.

Results

The table is saved with the new column attributes.

See also

- “Object name considerations” [SQL Anywhere Server - Database Administration]
- “UltraLite SQL data types” on page 275
- “Column data type considerations” [SQL Anywhere Server - Database Administration]
- “ALTER TABLE statement [UltraLite]” on page 401

Deleting UltraLite tables

You can delete tables when you no longer need them.

Prerequisites

You can drop any table if the table:

- Is not being used as an article in a publication.
- Does not have any columns that are referenced by another table's foreign key.

In these cases, you must change the publication or delete the foreign key before you can successfully delete the table.

Task

1. Using Sybase Central, connect to the UltraLite database.
2. In the left pane, double-click Tables.
3. Right-click a table and click Delete.
4. Click Yes.

Results

The table is deleted.
**See also**

- “DROP TABLE statement [UltraLite]” on page 421

---

**Browsing the information in UltraLite tables**

View the data held within the tables of an UltraLite database.

**Prerequisites**

The database must be connected and selected.

**Context and remarks**

Tables can be user tables or system tables. You can filter tables by showing and hiding system tables from your current view of the database. Because UltraLite does not have a concept of ownership, all users can browse all tables.

**Task**

1. Using Sybase Central, connect to the UltraLite database.
2. To view a list of tables, double-click **Tables**.
3. To view table data, double-click a table and click the **Data** tab in the right pane.

**Results**

The tables and data are displayed.

**See also**

- “UltraLite system tables” on page 220

---

**Data copying and pasting to or from UltraLite databases**

With Sybase Central you can copy and paste and drag and drop. This data transferral allows you to share or move objects among one or more databases. By copying and pasting or dragging and dropping you can share data as described by the table that follows.

<table>
<thead>
<tr>
<th>Target</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Another UltraLite or SQL Anywhere database.</td>
<td>A new object is created, and the original object's code is copied to the new object.</td>
</tr>
<tr>
<td>The same UltraLite database.</td>
<td>A copy of the object is created; you must rename the new object.</td>
</tr>
</tbody>
</table>
Note
You can copy data from a database opened in MobiLink and paste it into an UltraLite database. However, you cannot paste UltraLite data into a database opened in MobiLink.

Sybase Central
When you copy any of the following objects in the UltraLite plug-in, the SQL for the object is also copied to the clipboard. You can paste this SQL into other applications, such as Interactive SQL or a text editor. For example, if you copy an index in Sybase Central and paste it into a text editor, the CREATE INDEX statement for that index appears. You can copy the following objects in the UltraLite plug-in:

- Articles
- Columns
- Foreign keys
- Indexes
- Publications
- Tables
- Unique constraints

Interactive SQL
With Interactive SQL you can also copy data from a result set into another object.

- Use the SELECT statement results into a named object.
- Use the INSERT statement to insert a row or selection of rows from elsewhere in the database into a table.

See also
- “INSERT statement [UltraLite]” on page 424
- “SELECT statement [UltraLite]” on page 429

Entity-relationship diagrams from the UltraLite plug-in
When you are connected to a database from the UltraLite plug-in, you can view an entity-relationship diagram of the tables in the database. When you have the database selected, click the ER Diagram tab in the right pane to see the diagram.
When you rearrange objects in the diagram, the changes persist between Sybase Central sessions. Double-clicking a table takes you to the column definitions for that table.

See also
● “Database creation” [SQL Anywhere Server - Database Administration]

UltraLite indexes

An index is a set of pointers to rows in a table based on the order of the values of data in one or more table columns. The index is a database object that is maintained automatically by UltraLite after it has been created. When UltraLite optimizes a query, it scans existing indexes to see if one exists for the table(s) named in the query. If it can help UltraLite return rows more quickly, the index is used. If you are using the UltraLite Table API in your application, you can specify an index that helps determine the order in which rows are traversed.

Tip
Indexes can improve the performance of a query—especially for large tables. To see whether a query is using a particular index, you can check the execution plan with Interactive SQL.

Alternatively, your UltraLite applications can include PreparedStatement objects which have a method to return plans.

UltraLite supports the following indexes. These indexes can be single or multi-column (also known as composite indexes). You cannot index LONG VARCHAR or LONG BINARY columns.
Index | Characteristics
--- | ---
Primary key | Required. An instance of a unique key. You can only have one primary key. Values in the indexed column or columns must be unique and cannot be NULL.
Foreign key | Optional. Values in the indexed column or columns can be duplicated. Nullability depends on whether the column was created to allow NULL. Values in the foreign key columns must exist in the table being referenced.
Unique key | Optional. Values in the indexed column or columns must be unique and cannot be NULL.
Non-unique index | Optional. Values in the indexed column or columns can be duplicated and can be NULL.
Unique index | Optional. Values in the indexed column or columns cannot be duplicated and can be NULL.

1 A foreign key can reference either a primary key or a unique key.

2 Also known as a unique constraint.

About composite indexes

Multi-column indexes are sometimes called composite indexes. Additional columns in an index can allow you to narrow down your search, but having a two-column index is not the same as having two separate indexes. For example, the following statement creates a two-column composite index:

```
CREATE INDEX name
ON Employees ( Surname, GivenName )
```

A composite index is useful if the first column alone does not provide high selectivity. For example, a composite index on Surname and GivenName is useful when many employees have the same surname. A composite index on EmployeeID and Surname would not be useful because each employee has a unique ID, so the column Surname does not provide any additional selectivity.

See also

- “Index scan creation and maintenance” on page 441
- “Execution plans in UltraLite” on page 446
- “Composite indexes” [SQL Anywhere Server - SQL Usage]
- UltraLite.NET: “Data creation and modification using the ULTable class” [UltraLite - .NET Programming]
- “ULCommand.Prepare method [UltraLite.NET]” [UltraLite - .NET Programming]
- UltraLite for C++: “Data creation and modification using the ULTable class” [UltraLite - C and C++ Programming]
- “ULPreparedStatement class [UltraLite C++]” [UltraLite - C and C++ Programming]
When to use an index

Use an index when:

- **You want UltraLite to maintain referential integrity**  An index also affords UltraLite a means of enforcing a uniqueness constraint on the rows in a table. You do not need to add an index for data that is very similar.

- **The performance of a particular query is important to your application**  If an index improves performance of a query and the performance of that query is important to your application and is used frequently, then you want to maintain that index. Unless the table in question is extremely small, indexes can improve search performance dramatically. Indexes are typically recommended whenever you search data frequently.

- **You have complicated queries**  More complicated queries, (for example, those with JOIN, GROUP BY, and ORDER BY clauses), can yield substantial improvements when an index is used—though it may be harder to determine the degree to which performance has been enhanced. Therefore, test your queries both with and without indexes, to see which yields better performance.

- **The size of an UltraLite table is large**  The average time to find a row increases with the size of the table. Therefore, to increase searchability in a very large table, consider using an index. An index allows UltraLite to find rows quickly—but only for columns that are indexed. Otherwise, UltraLite must search every row in the table to see if the row matches the search condition, which can be time consuming in a large table.

- **The UltraLite client application is not performing a large amount of insert, update, or delete operations**  Because UltraLite maintains indexes along with the data itself, an index in this context will have an adverse effect on the performance of database operations. For this reason, you should restrict the use of indexes to data that will be queried regularly as described in the point above. Maintaining the UltraLite default indexes (indexes for primary keys and for unique constraints) may be enough.

- **Use indexes on columns involved in WHERE clauses and/or ORDER BY clause**  These indexes can speed the evaluation of these clauses. In particular, an index helps optimize a multi-column ORDER BY clause—but only when the placement of columns in the index and ORDER BY clauses are exactly the same.

**Index types**

UltraLite supports different types of indexes: unique keys, unique indexes, and non-unique indexes. What differentiates one from the others is what is allowed in that index.

<table>
<thead>
<tr>
<th>Index characteristic</th>
<th>Unique keys</th>
<th>Unique indexes</th>
<th>Non-unique indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows duplicate index entries for rows that have the same values in indexed columns.</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
### Index characteristic

<table>
<thead>
<tr>
<th></th>
<th>Unique keys</th>
<th>Unique indexes</th>
<th>Non-unique indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows null values in index columns.</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Note**

You can create foreign keys to unique keys, but not to unique indexes.

Also, manually creating an index on a key column is not necessary and generally not recommended. UltraLite creates and maintains indexes for unique keys automatically.

**See also**

- “Adding an UltraLite index” on page 55

## Adding an UltraLite index

Adding indexes to databases speeds up the search process.

### Prerequisites

The database must be connected.

### Context and remarks

**Note**

UltraLite does not detect duplicate or redundant indexes. As indexes must be maintained with the data in your database, add your indexes carefully.

### Task

1. Using Sybase Central, connect to the UltraLite database.
2. Right-click **Indexes**, and click **New » Index**.
3. Follow the instructions in the wizard.

### Results

The index is created.

### Example

To speed up a search on employee surnames in a database that tracks employee information, and tune the performance of queries against this index, you could create an index called EmployeeNames and increase the hash size to 20 bytes with the following statement:
CREATE INDEX EmployeeNames
ON Employees (Surname, GivenName)
WITH MAX HASH SIZE 20

This statement creates an index with the default maximum hash size you have configured. To create an index that overrides the default, ensure you use the WITH MAX HASH SIZE value clause to set a new value for this index instance.

See also
● “CREATE INDEX statement [UltraLite]” on page 407

Dropping an UltraLite index

Dropping an index deletes it from the database.

Prerequisites

There are no prerequisites for this task.

Task

1. Using Sybase Central, connect to the UltraLite database.
2. In the left pane, double-click Indexes.
3. Right-click an index and then click Delete.
4. Click Yes.

Results

The index is removed from the database.

See also
● “DROP INDEX statement [UltraLite]” on page 419

UltraLite users

A typical UltraLite database contains one user ID and password. UltraLite databases are created with a default user ID of DBA and default password of sql unless otherwise specified. UltraLite Java edition databases are created with a default password of DBA.

Changing the user schema is optional and not required. Many applications do not need database-level authentication and assume that a device level password is sufficient authentication to access an application and its data.

Common reasons for not authenticating users may be because the deployment is to a single-user device, or that it is too awkward to prompt a user each time they start the application.
You do not need to include a user ID or password in the database connection string if you do not need database-level authentication. The simplest UltraLite connection string is `DBF=filename`. The remainder of this section explains how UltraLite implements user IDs and describes how to use them for cases where you need explicit user authentication.

When developing an UltraLite application with a custom user authentication interface, you can effectively use the UltraLite user IDs and password hashes stored in an UltraLite database to validate user-supplied credentials and avoid creating your own password hashing algorithm. By adding users to your UltraLite database, you store their user IDs and password hashes. You can then validate the user-supplied credentials in your application by attempting to connect to the database with the UID and PWD connection parameters, where `UID=username` and `PWD=password`. A successful UltraLite database connection indicates that the user is authentic.

### Caution

Unlike SQL Anywhere users, UltraLite users are created and managed solely for authentication and not for object ownership or specific database roles and privileges. Once users are authenticated, they gain full access to the database.

By creating user IDs and passwords, you control connections to the UltraLite database but do not secure the data in the database file. The contents are stored as plain text and can be read directly.

To secure the database contents, it is recommended that you encrypt the file. When you encrypt the file, you can authenticate users with an encryption key rather than a user ID and password.

You can obfuscate the file to alter the storage so that data is not stored as plain text, but this approach does not secure the data.

For more information, see "Database security" on page 26 and "UltraLite DBKEY connection parameter" on page 166.

### Note

UltraLite user IDs are different from MobiLink user names.

### Limitations

User IDs and passwords are not stored in UltraLite Java edition databases. Those databases are protected with a single password and do not require user creation or authentication. For more information, see "UltraLite and UltraLite Java edition database creation and connection approaches" [UltraLite - Java Programming].

The following limitations apply to UltraLite user IDs:

- UltraLite supports up to four unique user IDs per UltraLite database.
- User IDs and passwords can be changed using Sybase Central, SQL statements, or UltraLite API methods in your application.
- User IDs have a 31-character limit.
UltraLite database schemas

- User IDs cannot include leading single quotes ('), leading double quotes ("), or semicolons (;).

- User IDs are always case insensitive and passwords are always case sensitive.

- User IDs cannot be renamed. You can only add new user IDs and delete existing ones from an existing database connection.

- Users cannot be listed programmatically using the UltraLite APIs. You can only use database tools to list existing users in the database.

- When connecting to an UltraLite database for the first time, the UID and PWD are the same values that were set when the database was created. UltraLite attempts to connect with the DBA user ID and sql password when these connection parameters are not specified. You do not need to supply a username or password when connecting to the database if you did not explicitly set a username and password during its creation.

See also

- “UltraLite UID connection parameter” on page 176
- “UltraLite PWD connection parameter” on page 172

Connection parameters for managing UltraLite users

You can use the UID and PWD connection parameters to create or authenticate users in an UltraLite database.

Note

As an alternative to connection parameters, you can use the following UltraLite API methods in your application to grant or revoke user access to an UltraLite database:

- “ULConnection.GrantConnectTo method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULConnection.RevokeConnectFrom method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULGrantConnectTo method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULRevokeConnectFrom method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULConnection.GrantConnectTo method [UltraLite.NET]” [UltraLite - .NET Programming]
- “ULConnection.RevokeConnectFrom method [UltraLite.NET]” [UltraLite - .NET Programming]

Grant and revoke methods are not available in the UltraLiteJ API. For Android smartphones, set the UID and PWD connection parameters using the ConfigPersistent.setConnectionString method.

Use connection parameters with UltraLite databases

For most UltraLite APIs, the createDatabase method of a DatabaseManager object can be used to create a new UltraLite database with the specified connection and creation parameters.

The following example illustrates how to create a default user for a new UltraLite database by passing the UID and PWD parameters to the CreateDatabase method in the UltraLite C++ API:
The following example illustrates how to authenticate a user in an existing UltraLite database by passing the **UID** and **PWD** parameters to the OpenConnection method in the UltraLite C++ API:

```cpp
ULConnection * conn;
ULError ulerr;
ULDatabaseManager::OpenConnection("dbf=sample.udb;uid=test-name;pwd=test-password", &ulerr);
```

**Use connection parameters with UltraLite databases on Android smartphones**

For the UltraLiteJ API, you use the setConnectionString method of a Configuration object in the UltraLiteJ API to create or authenticate users.

The following example illustrates how to create a default user for a new UltraLite database by passing the **UID** and **PWD** parameters to the createDatabase method in the UltraLiteJ API:

```java
ConfigFile config =
  DatabaseManager.createConfigurationFileAndroid("DBname.udb",
  getApplicationContext());
config.setConnectionString("uid=default-name;pwd=default-password");
Connection conn = DatabaseManager.createDatabase(config);
```

The following example illustrates how to authenticate a user in an existing UltraLite database by passing the **UID** and **PWD** parameters and the connect method in the UltraLiteJ API:

```java
ConfigFile config =
  DatabaseManager.createConfigurationFileAndroid("DBname.udb",
  getApplicationContext());
config.setConnectionString("uid=test-name;pwd=test-password");
Connection conn = DatabaseManager.connect(config);
```

As an alternative to the setConnectionString method, you can use the setPassword or setUserName methods to create or authenticate a user, respectively.

**Creating or authenticating users on an UltraLite Java edition database**

UltraLite Java edition databases do not support user authentication. Instead, you set a database password using the Configuration.setPassword method.

The following example illustrates how to set the password for an existing UltraLite Java edition database:

```java
ConfigNonPersistent config =
  DatabaseManager.createConfigurationNonPersistent("DBname.ulj");
config.setPassword("my_password");
Connection conn = DatabaseManager.connect(config);
```
See also

- “ConfigPersistent.setUserName method [Android] [UltraLiteJ]” [UltraLite - Java Programming]
- “Configuration.setPassword method [UltraLiteJ]” [UltraLite - Java Programming]
- “UltraLite UID connection parameter” on page 176
- “UltraLite PWD connection parameter” on page 172
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “ConfigPersistent.setConnectionString method [Android] [UltraLiteJ]” [UltraLite - Java Programming]
- “DatabaseManager.createDatabase method [UltraLiteJ]” [UltraLite - Java Programming]

SQL statements for managing UltraLite users

You can use the CREATE USER, ALTER USER, and DROP USER statements to manage users in an UltraLite database.

You cannot use these SQL statements with UltraLite Java edition databases.

Note
As an alternative to SQL statements, you can use the following UltraLite API methods in your application to grant or revoke user access to an UltraLite database:

- “ULConnection.GrantConnectTo method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULConnection.RevokeConnectFrom method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULGrantConnectTo method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULRevokeConnectFrom method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULConnection.GrantConnectTo method [UltraLite.NET]” [UltraLite - .NET Programming]
- “ULConnection.RevokeConnectFrom method [UltraLite.NET]” [UltraLite - .NET Programming]

Grant and revoke methods are not available in the UltraLiteJ API. For Android smartphones, construct a CREATE USER, ALTER USER, or DROP USER statement as a string variable and pass it to the Connection.prepareStatement method.

Example
The following example illustrates how to use the UltraLiteJ API connect to an existing UltraLite database on an Android smartphone, and use the CREATE USER statement to create a new user:

```java
ConfigFile config = DatabaseManager.createConfigurationFileAndroid("DBname.udb", getApplicationContext);
Connection conn = DatabaseManager.connect(config);

String sql_string = "CREATE USER test-user IDENTIFIED BY test-password";
PreparedStatement authenticator = conn.prepareStatement(sql_string);
authenticator.execute();
authenticator.close();
```
Creating an UltraLite user with Sybase Central

Use Sybase Central to create users for an UltraLite database.

Prerequisites

There are no prerequisites for this task.

Context and remarks

Sybase Central is not compatible with UltraLite Java edition databases.

As an alternative to Sybase Central, you can use the following UltraLite API methods in your application to grant user access to an UltraLite database:

- “ULConnection.GrantConnectTo method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULGrantConnectTo method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULConnection.GrantConnectTo method [UltraLite.NET]” [UltraLite - .NET Programming]

Grant methods are not available in the UltraLiteJ API. For Android smartphones, construct a CREATE USER statement as a string variable and pass it to the Connection.prepareStatement method.

Task

1. Using Sybase Central, connect to the UltraLite database.
2. Right-click the Users folder, and click New » User.
3. Follow the instructions in the wizard.

Results

The new user is created.
Deleting an UltraLite user with Sybase Central

Use Sybase Central to explicitly delete users from an UltraLite database.

Prerequisites

There are no prerequisites for this task.

Context and remarks

Sybase Central is not compatible with UltraLite Java edition databases.

Note

As an alternative to Sybase Central, you can use the following UltraLite API methods in your application to revoke user access to an UltraLite database:

- “ULConnection.RevokeConnectFrom method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULRevokeConnectFrom method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULConnection.RevokeConnectFrom method [UltraLite.NET]” [UltraLite - .NET Programming]

Revoke methods are not available in the UltraLiteJ API. Construct a DROP USER statement as a string variable and pass it to the Connection.prepareStatement method.

Task

1. Using Sybase Central, connect to the UltraLite database.
2. In the left pane, double-click the Users folder.
3. Right-click the user and click Delete.

Results

The user is deleted from the database.

See also

- “DROP USER statement [UltraLite]” on page 422
UltraLite as a MobiLink client

This section contains material that describes how to set up and run UltraLite clients for MobiLink synchronization.

UltraLite clients

Synchronizing an UltraLite database requires your application to set synchronization parameters identifying the address of the MobiLink server and other required information, and calling a synchronization function or executing the SYNCHRONIZE SQL statement. The option you chose depends on the API you are using.

UltraLite client synchronization behavior customization

Adding custom synchronization support to UltraLite can involve several tasks:

- **Maintain primary key uniqueness in synchronization models that include more than one remote client** Required. In a synchronization system, the primary key is the only way to identify the same row in different databases (remote and consolidated) and the only way to detect conflicts. Therefore, multiple clients must adhere to the following rules:
  - Every table that is to be synchronized must have a primary key.
  - Never update the values of primary keys.
  - Primary keys must be unique across all synchronized databases.

- **Ensure your date columns are set up so that fractional data is not lost** For a SQL Anywhere consolidated database this is not typically an issue. However, for databases like Oracle, there may be compatibility issues that you need to consider. For example, UltraLite and Oracle databases must share the same timestamp precision. Additionally, you should also add a TIMESTAMP to the Oracle database to avoid losing fractional second data when the UltraLite remote databases uploads data to the consolidated database.

- **Describe what data subsets you want to upload to the consolidated database** Optional. You only need to do this when you do not want to synchronize all data by default. To target what data you want to synchronize, use one or more subsetting techniques.

  For example, you may want to create a publication for high-priority data. The application could then synchronize this data over wireless networks. Because wireless networks can have high usage costs associated with them, you may want to limit these usage fees to those that are business critical. You can then synchronize less time-sensitive data from a cradle at a later time.

- **Initialize synchronization from your UltraLite application and supply the parameters that describe the session** Required. Programming synchronization has two parts: describing the session, and then initiating the synchronization operation.

  Describing the session primarily involves choosing a synchronization communication stream (also known as a network protocol), and the parameters for that stream, setting the version of your
synchronization scripts, and identifying the MobiLink user. However, there are other parameters you can set: for example, use the upload_only and download_only parameters to change the default bi-directional synchronization to one-way only.

All other important synchronization behaviors are controlled at the MobiLink server with MobiLink synchronization scripts. These include:

- What data is downloaded as updates or inserts to tables in the UltraLite remote.
- What processing is required on uploaded changes from a remote database.

You can write your synchronization scripts so that data is partitioned among remote databases in an appropriate manner.

See also

- “Unique primary keys” [MobiLink - Server Administration]
- “Primary key uniqueness in UltraLite” on page 64
- “Oracle consolidated database” [MobiLink - Server Administration]
- “UltraLite precision creation parameter” on page 146
- “UltraLite client synchronization design” on page 68
- “Synchronization setup for your UltraLite application” on page 74
- “MobiLink consolidated databases” [MobiLink - Server Administration]
- “Synchronization script writing” [MobiLink - Server Administration]
- “Direct row handling” [MobiLink - Server Administration]
- “Partitioned rows among remote databases” [MobiLink - Server Administration]

Primary key uniqueness in UltraLite

UltraLite can maintain primary key uniqueness using any of the techniques supported by MobiLink.

One of these methods is to use a GLOBAL AUTOINCREMENT column. GLOBAL AUTOINCREMENT is similar to AUTOINCREMENT, except that the domain is partitioned. UltraLite supplies column values only from the partition assigned to the database's global database ID. Each UltraLite database is assigned a unique integer global database ID.

A second method is to use a UUID primary key column. A UUID requires more data, but needs no distinct database identifier.

See also

- “Unique primary keys” [MobiLink - Server Administration]
- “UltraLite global_database_id option” on page 184

GLOBAL AUTOINCREMENT columns in UltraLite

You can declare the default value of a column in an UltraLite database to be of type GLOBAL AUTOINCREMENT. Before you can autoincrement these column IDs, you must first set the global database ID for the UltraLite database.
Caution
GLOBAL AUTOINCREMENT column values downloaded via MobiLink synchronization do not update the GLOBAL AUTOINCREMENT value counter. As a result, an error can occur should one MobiLink client insert a value into another client’s partition. To avoid this problem, ensure that each copy of your UltraLite application inserts values only in its own partition.

The global_database_id database option allows you to set the value in your UltraLite database. When deploying UltraLite, you must assign a different identification number to each database.

Allow UltraLite to supply default values for the column using the partition uniquely identified by the UltraLite database's number.

UltraLite follows these rules:

- If the column contains no values in the current partition, the first default value is $pn + 1$. $p$ represents the partition size and $n$ represents the global ID number.

- If the column contains values in the current partition, but all are less than $p(n + 1)$, the next default value will be one greater than the previous maximum value in this range.

- Default column values are not affected by values in the column outside the current partition; that is, by numbers less than $pn + 1$ or greater than $p(n + 1)$. Such values may be present if they have been replicated from another database via MobiLink synchronization.

For example, if you assigned your UltraLite database a global ID of 1 and the partition size is 1000, then the default values in that database would be chosen in the range 1001-2000. Another copy of the database, assigned the identification number 2, would supply default values for the same column in the range 2001-3000.

- Because you cannot set the global ID number to a negative value, the GLOBAL AUTOINCREMENT column values are always positive. The maximum identification number is restricted only by the column data type and the partition size.

- If you do not set a global ID value, or if you exhaust values from the partition, a NULL value is inserted into the column. Should NULL values not be permitted, the attempt to insert the row causes an error.

If you exhaust or will soon exhaust available values for columns declared as GLOBAL AUTOINCREMENT, you need to set a new global database ID. UltraLite chooses GLOBAL AUTOINCREMENT values from the partition identified by the global ID number, but only until the maximum value is reached. If you exceed values, UltraLite begins to generate NULL values. By assigning a new global database ID number, you allow UltraLite to set appropriate values from another partition.

One approach of choosing a new global database ID is to maintain a pool of unused global database ID values. This pool is maintained in the same manner as a pool of primary keys.
Tip
UltraLite APIs provide means of obtaining the proportion of numbers that have been used. The return value is a SHORT in the range 0-100 that represents the percent of values used so far. For example, a value of 99 indicates that very few unused values remain and the database should be assigned a new identification number. The method of setting this identification number varies according to the programming interface you are using.

See also
- “UltraLite global_database_id option” on page 184
- “Primary key pools” [SQL Remote]
- “Partition sizes” on page 67
- “UltraLite global_database_id option” on page 184
- “ULCommand.Connection property [UltraLite.NET]” [UltraLite - .NET Programming]
- “ULConnection.Synchronize method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULSetDatabaseID method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULGlobalAutoincUsage method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “Connection.setDatabaseId method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.getDatabaseId method [BlackBerry] [UltraLiteJ]” [UltraLite - Java Programming]

Methods for finding the last assigned GLOBAL AUTOINCREMENT value
You can retrieve the GLOBAL AUTOINCREMENT value that was chosen during the most recent insert operation. Since these values are often used for primary keys, knowing the generated value may let you more easily insert rows that reference the primary key of the first row. You can check the value with:

- **UltraLite for C/C++** Use the GetLastIdentity function on the ULConnection object.
- **UltraLite.NET** Use the LastIdentity property on the ULConnection class.
- **API UltraLiteJ** Use the getLastIdentity method on the Connection interface.

The returned value is an unsigned 64-bit integer, database data type UNSIGNED BIGINT. Since this statement only allows you to determine the most recently assigned default value, you should retrieve this value soon after executing the insert statement to avoid spurious results.

Note
Occasionally, a single INSERT statement may include more than one column of type GLOBAL AUTOINCREMENT. In this case, the return value is one of the generated default values, but there is no reliable means to determine which one. For this reason, you should design your database and write your INSERT statements in a way that avoids this situation.
Partition sizes

The partition size is 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.

For columns of type INT or UNSIGNED INT, the default partition size is \(2^{16} = 65536\); for columns of other types the default partition size is \(2^{32} = 4294967296\). Since these defaults may be inappropriate, it is best to specify the partition size explicitly.

Default partition sizes for some data types are different in UltraLite applications than in SQL Anywhere databases. Declare the partition size explicitly if you want different databases to remain consistent.

Overriding the partition size for a GLOBAL AUTOINCREMENT column

Increasing the partition size of a GLOBAL AUTOINCREMENT column ensures that the supply of numbers within the partition is rarely exhausted.

Prerequisites

You must be connected to an UltraLite database.

Task

1. Create a table with a GLOBAL AUTOINCREMENT column that has a partition size specified in parentheses.

   Execute the following SQL code:

   ```sql
   CREATE TABLE customer (  
       id    INT DEFAULT GLOBAL AUTOINCREMENT (5000),  
       name  VARCHAR(128) NOT NULL,  
       PRIMARY KEY (id)
   )
   ```
A simple reference table with two columns—an integer that holds a customer identification number, and a character string that holds the customer's name is created. The ID has a partition size of 5000.

2. Connect to the UltraLite database in Sybase Central.

3. Right-click the ID column of the customer table and click Properties.

4. Click the Value tab.

5. Enter a positive integer that is greater than 5000 in the Partition Size field.

**Results**

The partition size of the ID column is updated according to the value entered in the Partition Size field.

**UltraLite client synchronization design**

All data in an UltraLite database is synchronized by default. If you are new to deploying UltraLite as a MobiLink remote database, plan to use the default behavior initially.

Once you become comfortable with the synchronization process, you may decide to customize the behavior of the synchronization operation to capture more complex business logic. Designing custom synchronization behavior requires that you ask the following questions. If your business requirements are simple, you may only need to use a single synchronization feature. However, in very complex deployments, you may need to use multiple synchronization features to configure the desired synchronization behavior.

<table>
<thead>
<tr>
<th>Design question</th>
<th>If you answer yes, use the following</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you want to download changes from the consolidated database but not have local changes uploaded to the database?</td>
<td>The download_only table name suffix allows you to identify any tables for which the synchronization should be download only. Changes made to the local tables are not uploaded to the consolidated database.</td>
</tr>
<tr>
<td>Do you want to exclude tables from synchronization?</td>
<td>The nosync table name suffix allows you to identify any tables that you do not want to synchronize.</td>
</tr>
<tr>
<td>Do you only want to synchronize entire tables even when data hasn't changed?</td>
<td>The allsync table name suffix allows you to synchronize the entire table, even when no changes are detected.</td>
</tr>
</tbody>
</table>
### Design question

<table>
<thead>
<tr>
<th>Do you want to synchronize an entire table or just rows that meet specific conditions? Does some of the data require synchronization priority due to its importance or time-sensitivity?</th>
<th>A publication includes articles that list the tables that require synchronization. An article can include a WHERE clause that specifies the rows to upload based on whether the rows meet the defined criteria. Multiple publications can address priority issues that require certain UltraLite data be uploaded before others.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you want a table order for synchronization because you have cycles of foreign keys?</td>
<td>The Table Order synchronization parameter allows you to determine the order of synchronization operations when you have foreign key cycles. However, foreign key cycles are generally not recommended for UltraLite.</td>
</tr>
</tbody>
</table>
| Do you want to control synchronization behavior? For example, do you need downloads to occur at the same time as uploads? Or do you want to change bi-directional synchronization to one-way only? | Use the appropriate synchronization parameter as part of:  
- Your application’s synchronization structure (or the synchronization enumeration).  
- The ulsync utility’s -e option. |
| Do you want your UltraLite client to be TLS-enabled? | What encryption algorithm you choose determines how your device must be set up according to the platform that runs on that device. |

### See also

- “UltraLite download-only tables” on page 70
- “UltraLite non-synchronizing tables” on page 69
- “UltraLite synchronize-all tables” on page 71
- “Publishing data in UltraLite” on page 72
- “Table order in UltraLite” on page 73
- “UltraLite synchronization parameters” on page 82
- “UltraLite network protocol options” on page 104
- “The synchronization process” [MobiLink - Getting Started]
- “The upload and the download” [MobiLink - Getting Started]

### UltraLite non-synchronizing tables

By creating the table using SYNCHRONIZE OFF, you control when to exclude the entire table from the upload operation. You can use these non-synchronizing tables for client-specific persistent data that is not required in the consolidated database. Other than being excluded from synchronization, you can use these tables in exactly the same way as other tables in the UltraLite database.
The synchronization type for a table can only be changed if it does not have any unsynchronized changes that need to be uploaded.

If you create a table with a _nosync suffix, you can only rename that table so it retains the _nosync suffix. For example, the following ALTER TABLE statement with a rename clause is not allowed because the new name no longer ends in nosync:

```
ALTER TABLE purchase_comments_nosync
RENAME comments
```

To correct this, the statement must be rewritten to include this suffix:

```
ALTER TABLE purchase_comments_nosync
RENAME comments_nosync
```

You can alternatively use publications to achieve the same effect.

As an alternative to creating or altering a table with the SYNCHRONIZE OFF clause, you can append the phrase _nosync to the table name to turn it into a non-synchronizing table.

**See also**

- “Download Only synchronization parameter” on page 88
- “Upload Only synchronization parameter” on page 101
- “Additional Parameters synchronization parameter” on page 83
- “CREATE TABLE statement [UltraLite]” on page 411
- “ALTER TABLE statement [UltraLite]” on page 401
- “Publications” [MobiLink - Client Administration]
- “UltraLite publications” on page 77
- “CREATE PUBLICATION statement [UltraLite]” on page 409
- “Synchronization script writing” [MobiLink - Server Administration]

**UltraLite download-only tables**

By creating the table using SYNCHRONIZE DOWNLOAD, you exclude entire tables from the upload operation. You can use these tables for data that should not synchronized to the consolidated database. Other than being excluded from synchronization, you can use these tables in exactly the same way as other tables in the UltraLite database.

You can alternatively use publications to achieve the same effect.

The synchronization type for a table can only be changed if it does not have any unsynchronized changes that need to be uploaded.
As an alternative to creating or altering a table with the SYNCHRONIZE DOWNLOAD clause, you can append the phrase `_download_only` to the table name to turn it into a download-only table.

See also

- “Download Only synchronization parameter” on page 88
- “Upload Only synchronization parameter” on page 101
- “Additional Parameters synchronization parameter” on page 83
- “CREATE TABLE statement [UltraLite]” on page 411
- “ALTER TABLE statement [UltraLite]” on page 401
- “Publications” [MobiLink - Client Administration]
- “UltraLite publications” on page 77
- “CREATE PUBLICATION statement [UltraLite]” on page 409
- “Synchronization script writing” [MobiLink - Server Administration]

**UltraLite synchronize-all tables**

By creating or altering a table using SYNCHRONIZE ALL you control whether to change the synchronization behavior during upload so that it synchronizes all table data, even if nothing has changed since the previous synchronization session.

Note

The synchronization type for a table can only be changed if it does not have any unsynchronized changes that need to be uploaded.

Some UltraLite applications require user/client-specific data that you can store in a SYNCHRONIZE ALL TABLES. You can upload the data in the table to a temporary table in the consolidated database, use the data to control synchronization by your other scripts without having the data maintained in the consolidated database. For example, you may want your UltraLite applications to indicate which channels or topics they are interested in, and use this information to download the appropriate rows.

Note

As an alternative to creating or altering a table with the SYNCHRONIZE ALL clause, you can append the phrase `_allsync` to the table name to turn it into a synchronize-all table.

See also

- “Download Only synchronization parameter” on page 88
- “Upload Only synchronization parameter” on page 101
- “Additional Parameters synchronization parameter” on page 83
- “CREATE TABLE statement [UltraLite]” on page 411
- “ALTER TABLE statement [UltraLite]” on page 401
- “Publications” [MobiLink - Client Administration]
- “UltraLite publications” on page 77
- “CREATE PUBLICATION statement [UltraLite]” on page 409
- “Synchronization script writing” [MobiLink - Server Administration]
Publishing data in UltraLite

Publications define a set of articles that describe the data to be synchronized. You can add publications to an UltraLite database using Sybase Central or SQL.

Prerequisites

There are no prerequisites for this task.

Context and remarks

Publication articles can be a whole table, or can define a subset of the data in a table. You can include an optional predicate (a WHERE clause) to define a subset of rows from a given table. Publications are more flexible than creating tables with SYNCHRONIZE OFF. To synchronize data subsets of an UltraLite database separately, use multiple publications. You can then combine publications with upload-only or download-only synchronization parameters to synchronize high-priority changes efficiently.

Note

The maximum number of user publications in UltraLite is 63.

UltraLite publications do not support the definition of column subsets, nor the SUBSCRIBE BY clause that is available in SQL Anywhere. If columns in an UltraLite table do not exactly match tables in a consolidated database, use MobiLink scripts to resolve those differences.

You do not need to set a table synchronization order in a publication. If table order is important for your deployment, you can set the table order when you synchronize the UltraLite database by setting the Table Order synchronization parameter.

Task

1. Connect to the UltraLite database using the UltraLite plug-in.

2. Right-click the Publications folder and click New » Publication.

3. Enter a name for the new publication. Click Next.

4. On the Tables tab, click a table in the Matching Tables list. Click Add.

   The table appears in the Selected Tables list on the right.

5. Add additional tables.

6. If necessary, click the Where tab to specify the rows to be included in the publication. You cannot specify column subsets.

7. Click Finish.

Results

The new publication is created.
Table order in UltraLite

By setting the Table Order synchronization parameter you can control the order of synchronization operations. To specify a table order for synchronization, you can use the TableOrder parameter programmatically or as part of the ulsync utility during testing. The TableOrder parameter specifies the order of tables that are to be uploaded.

You only need to explicitly set the table order if your UltraLite database has:

- Foreign key cycles. You must then list all tables that are part of a cycle.

- Different foreign key relationships from those used in the consolidated database.

Avoiding synchronization issues with foreign key cycles

Table order is particularly important for UltraLite databases that use foreign key cycles. A cycle occurs when you link a series of tables together such that a circle is formed. However, due to complexities that arise when cycles between the consolidated database and the UltraLite remote database differ, foreign key cycles are not recommended.

With foreign key cycles, you should order your tables so that operations for a primary table come before the associated foreign table. A Table Order parameter ensures that the insert in the foreign table will have its foreign key referential integrity constraint satisfied (likewise for other operations like delete).

In addition to table ordering, another method you can use to avoid synchronization issues is to postpone the checking of referential integrity until the transaction is committed. If your consolidated database is a SQL Anywhere database, set one of the foreign keys to `check on commit`. This ensures that foreign key referential integrity is checked during the commit phase rather than when the operation is initiated. For example:

```sql
CREATE TABLE c (
   id INTEGER NOT NULL PRIMARY KEY,
   c_pk INTEGER NOT NULL
);  
CREATE TABLE p (
   pk INTEGER NOT NULL PRIMARY KEY,
   c_id INTEGER NOT NULL,
   FOREIGN KEY p_to_c (c_id) REFERENCES c(id)
);  
ALTER TABLE c 
   ADD FOREIGN KEY c_to_p (c_pk) REFERENCES p(pk) 
   CHECK ON COMMIT;
```
If your consolidated database is from another database vendor, check to see if the database has similar methods of checking referential integrity. If so, you should implement this method. Otherwise, you must redesign table relationships to eliminate all foreign key cycles.

See also

- “Additional Parameters synchronization parameter” on page 83
- “Referential integrity and synchronization” [MobiLink - Getting Started]

Synchronization setup for your UltraLite application

In UltraLite, synchronization begins by opening a specific connection with the MobiLink server over the configured communication stream (also known as a network protocol). In addition to synchronization support for direct network connections, Windows Mobile devices also support ActiveSync synchronization.

Defining the connection

Each UltraLite remote database that synchronizes with a MobiLink server does so over a network protocol. You set the network protocol with the synchronization stream parameter. Supported network protocols include TCP/IP, HTTP, HTTPS, and TLS. For the protocol you choose, you also need to supply stream parameters that define other required connection information like the MobiLink server host and the port. You must also supply the MobiLink user information and the synchronization script version.

Defining the synchronization behavior

You can control synchronization behavior by setting various synchronization parameters. The way you set parameters depends on the specific UltraLite interface you are using.

Important behaviors to consider include:

- **Synchronization direction** By default, synchronization is bi-directional. When using only one-way synchronizations, remember to use the appropriate upload_only or download_only parameter. By performing one-way synchronizations, you minimize the synchronization time required. Also, with download-only synchronization, you do not have to commit all changes to the UltraLite database before synchronization. Uncommitted changes to tables not involved in synchronization are not uploaded, so incomplete transactions do not cause problems.

  To use download-only synchronization, you must ensure that rows overlapping with the download are not changed locally. If any data is changed locally, synchronization fails in the UltraLite application with a SQLE_DOWNLOAD_CONFLICT error.

- **Concurrent changes during synchronization** During the upload phase, UltraLite applications can access UltraLite databases in a read-only fashion. During the download phase, read-write access is permitted, but if an application changes a row that the download then attempts to change, the download will fail and roll back. You can disable concurrent access to data during synchronization by setting the DisableConcurrency synchronization parameter.

The following procedure is generally used to add synchronization functionality to your application:
1. Supply the necessary synchronization parameters and protocol options needed for the session as fields of a synchronization information structure.

For example, using the UltraLite C/C++ API, you add synchronization to your application by setting appropriate values in the ul_sync_info structure:

```c
ul_sync_info info;
// define a sync structure named "info"
ULEnableTcpipSynchronization( &sqlca );
// use a TCP/IP stream
conn->InitSynchInfo( &info );
// initialize the structure
info.stream = ULSocketStream();
// specify the Socket Stream
info.stream_parms = UL_TEXT( "host=myMLserver;port=2439" );
// set the MobiLink host information
info.version = UL_TEXT( "custdb 11.0" );
// set the MobiLink version information
info.user_name = UL_TEXT( "50" );
// set the MobiLink user name
info.download_only = ul_true;
// make the synchronization download-only
```

2. Initialize synchronization.

For direct synchronization, you would call an API-specific synchronization method. These methods return a boolean value, indicating success or failure of the synchronization operation. If the synchronization fails, you can examine detailed error status fields in another structure to get additional error information.

For ActiveSync synchronization, you must catch the synchronization message from the ActiveSync provider and use the DoSync method to call the ULSynchronize method.

3. Use an observer callback function to report the progress of the synchronization to the user.

**Tip**

If you have an environment where DLLs fail either because the DLL is very large or the network connection is unreliable, you may want to implement resumable downloads. See “Failed downloads” [MobiLink - Server Administration] and “Resumption of failed downloads” [MobiLink - Server Administration].
See also

- “ActiveSync with UltraLite on Windows Mobile” on page 81
- “Upload-only and download-only synchronizations” [MobiLink - Server Administration]
- “The upload and the download” [MobiLink - Getting Started]
- “UltraLite synchronization parameters” on page 82
- “UltraLite network protocol options” on page 104
- UltraLite.NET: “MobiLink data synchronization” [UltraLite - .NET Programming]
- UltraLite C/C++: “MobiLink data synchronization” [UltraLite - C and C++ Programming]
- UltraLite C/C++: “ActiveSync synchronization setup” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “Synchronization setup for an embedded SQL application” [UltraLite - C and C++ Programming]
- UltraLiteJ API: “MobiLink data synchronization” [UltraLite - Java Programming]

MobiLink file transfers

UltraLite supports the ability to transfer files with the MobiLink server.

For all other APIs, use the MobiLink file transfer mechanism when:

- You have multiple files that you need to deploy to multiple devices, particularly when corporate firewalls are used as a security measure. Because MobiLink is already configured to handle synchronization through these firewalls, the MLFileTransfer mechanism makes device provisioning for upgrades and other types of file transfers very convenient.

- You have files that you want to target to a specific MobiLink user ID. This requires that you create one or more user-specific directories on the MobiLink server for each user ID. Otherwise, if you only have a single version of the file, you can use a default directory.

How file transfers work

You can employ one of two MobiLink-initiated file transfer mechanisms to download files to a device: run the mlfiletransfer utility for desktop transfers, or call the appropriate function for the API you are using to code your UltraLite application. Both approaches require that you:

1. Describe the transfer destination.

   Whether you use the mlfiletransfer utility from the desktop, or whether you use the function appropriate to your API, you must set the local path and file name of the file on the target device or desktop computer. If none are supplied in the application or by the end user, then the source file name is assumed and the file is stored in the current directory.

   The destination directory of the target can vary depending on the device's operating system:

   - On Windows Mobile, if the destination is NULL, the file is stored in the root directory (\).
   
     The file name must follow file name conventions for Windows Mobile. See “Windows Mobile” on page 34.

   - On the desktop, if the destination is NULL, the file is stored in the current directory.
The file name must follow file name conventions for the desktop system. See “Desktop” on page 34.

- On iPhone, you should store files in your application’s document directory. You can get the location of the document directory by calling the NSSearchPathForDirectoriesInDomains using the NSDocumentDirectory parameter.

- On BlackBerry smartphones, set the location using the FileTransfer object.

2. Set the MobiLink user credentials that allow the user to be identified and the correct file(s) to be downloaded.

This user name and password are separate from any database user ID and password, and serve to identify and authenticate the application to the MobiLink server.

3. Set the stream type you want to use, and define the parameters for the desired stream. These are the same parameters supported by UltraLite for MobiLink synchronization.

Most synchronization streams require parameters to identify the MobiLink server address and control other behavior. If you set the stream type to a value that is invalid for the platform, the stream type is set to TCP/IP, except for UltraLite Java edition databases which supports only HTTP.

4. Describe the required behavior for the transfer mechanism.

For example, you can set properties that allow this mechanism to force a download even when the file already exists on the target and has not changed, or that allow partial downloads to be resumed. You can also set whether you want the progress to be monitored and reported upon.

5. Ensure the MobiLink server is running and has been started with the -ftr option.

6. Start the transfer, and, if applicable, monitor the download progress.

By displaying the download progress, the user can cancel and resume the download at a later time.

See also

- “UltraLite synchronization parameters” on page 82
- “UltraLite network protocol options” on page 104
- “-ftr mlsv16 option” [MobiLink - Server Administration]
- “MobiLink File Transfer utility (mlfiletransfer)” [MobiLink - Client Administration]
- “MLFileDownload method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “MLFileUpload method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULFileTransfer class [UltraLite.NET]” [UltraLite - .NET Programming]
- “FileTransfer interface [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite publications

A publication is a database object that identifies the data that is to be synchronized. To synchronize all tables and all rows of those tables in your UltraLite database, do not create any publications.
A publication consists of a set of articles. Each article may be an entire table, or may be rows in a table. You can define this set of rows with a WHERE clause.

Each database can have multiple publications, depending on the desired synchronization logic. For example, you may want to create a publication for high-priority data. The user can synchronize this data over high-speed wireless networks. Because wireless networks can have usage costs associated with them, you would want to limit these usage fees to those that are business-critical only. All other less time-sensitive data could be synchronized from a cradle at a later time.

You create publications using Sybase Central or with the CREATE PUBLICATION statement. In Sybase Central, all publications and articles appear in the Publications folder.

Usage notes

- UltraLite publications do not support the definition of column subsets, nor the SUBSCRIBE BY clause. If columns in an UltraLite table do not exactly match tables in a SQL Anywhere consolidated database, use MobiLink scripts to resolve those differences.

- Columns are always sent in the order in which they were defined in the CREATE TABLE statement.

- You do not need to set a table synchronization order in a publication. If table order is important for your deployment, you can set the table order when you synchronize the UltraLite database by setting the Table Order synchronization parameter.

- Because object ownership is not supported in UltraLite, any user can delete a publication.

See also

- “Table order in UltraLite” on page 73
- “Publications” [MobiLink - Client Administration]
- “UltraLite client synchronization design” on page 68
- “Synchronization script writing” [MobiLink - Server Administration]

Publishing whole tables in UltraLite

A publication consists of a set of articles. The simplest publication you can make consists of a single article, which consists of all rows and columns of a table.

Prerequisites

There are no prerequisites for this task.

Context and remarks

You can use either Sybase Central or Interactive SQL to perform this task.

Task

1. Using Sybase Central, connect to the UltraLite database.
2. Right-click the **Publications** folder, and click **New » Publication**.

3. In the **What Do You Want To Name The New Publication** field, type a name for the new publication. Click **Next**.

4. On the **Tables** tab, click tables in the **Available Tables** list. Click **Add**.

5. Click **Finish**.

**Results**

The publication is created.

**Next**

None.

**See also**

- “CREATE PUBLICATION statement [UltraLite]” on page 409
- “UltraLite clients” on page 63

---

**Publishing a subset of rows from an UltraLite table**

A publication can only contain specific table rows. A WHERE clause limits the rows that are uploaded to those that have changed and satisfy a search condition in the WHERE clause.

**Prerequisites**

There are no prerequisites for this task.

**Context and remarks**

To upload all changed rows, do not specify a WHERE clause.

**Task**

1. Using Sybase Central, connect to the UltraLite database.

2. Right-click the **Publications** folder, and click **New » Publication**.

3. In the **What Do You Want To Name The New Publication** field, type a name for the new publication.

4. Click **Next**.

5. In the **Available Tables** list, click a table and click **Add**.

6. Click the **WHERE Clauses** tab, and click the table from the **Articles** list. Optionally, you can use the **Insert** window to assist you in formatting the search condition.
7. Click Finish.

Results

The rows that are uploaded are now limited to those that have changed and that satisfy the search condition in the WHERE clause.

Next

None.

See also

● “CREATE PUBLICATION statement [UltraLite]” on page 409
● “UltraLite clients” on page 63

Dropping a publication for UltraLite

Dropping a table's publications allows you to synchronize all the tables and rows of that table in your UltraLite database.

Prerequisites

There are no prerequisites for this task.

Context and remarks

You can drop a publication using either Sybase Central or Interactive SQL.

Task

1. In Sybase Central, connect to the UltraLite database.

2. In the left pane, double-click the Publications folder.

3. Right-click the publication and click Delete.

4. Click Yes.

Results

The publication is deleted.

See also

● “DROP PUBLICATION statement [UltraLite]” on page 420
● “UltraLite clients” on page 63
ActiveSync with UltraLite on Windows Mobile

While you can synchronize data from a Windows Mobile device over an Ethernet or Wi-Fi connection, this section describes how to configure your desktop and device to use ActiveSync synchronization so that your UltraLite database is synchronized at the same time as other ActiveSync operations. To synchronize directly using one of the other alternative methods, you need to program your application to do so using an appropriate synchronize function.

To use ActiveSync initiated synchronization requires that you:

- Register all applications that need to use ActiveSync initiated synchronization with ActiveSync.
- Have the ActiveSync provider installed on your desktop, and deployed to your device.
  
  To determine which platforms the provider is supported on, see http://www.sybase.com/detail?id=1002288.

The ActiveSync architecture

The following diagram shows the computing layers required by the ActiveSync architecture.

![ActiveSync architecture diagram]

Notice that you must install the ActiveSync provider on your device in addition to your desktop. You can only have a single ActiveSync provider on a single computer. However, if you have more than one UltraLite application installed on a Windows Mobile device, you can register them with the same provider so they are synchronized simultaneously.

See also

- UltraLite C/C++: “ActiveSync synchronization setup” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “Synchronization setup for an embedded SQL application” [UltraLite - C and C++ Programming]
ActiveSync synchronization overview

1. ActiveSync begins a synchronization session.

2. The ActiveSync provider sends a synchronize notification message to the first registered application on the device. The application is started if it is not yet running.

3. WndProc is invoked for each registered application.

4. Once the application has determined that this is the synchronize notification message from ActiveSync, the application calls ULIsSynchronizeMessage to invoke the database synchronization procedure.

5. Once synchronization is complete, the application calls ULSignalSyncIsComplete to let the provider know that it has finished synchronizing.

6. Steps two-five are repeated for each application that has been registered with the provider.

UltraLite synchronization parameters

Synchronization parameters control the synchronization between an UltraLite database and the MobiLink server. The way you set parameters depends on the specific UltraLite interface you are using. This section describes the effects of the parameters, and provides links to other locations for information about to set them.

Note
The parameters described in this section only apply to UltraLite remote databases. To synchronize SQL Anywhere remote databases, see “MobiLink SQL Anywhere client utility (dbmlsync)” [MobiLink - Client Administration].

Required parameters

The following parameters are required:

- Stream Type
- User Name
- Version

The synchronization function throws an exception, such as SQLCode.SQLE_SYNC_INFO_INVALID or its equivalent, if you do not set these parameters.

Conflicting parameters

You can specify at most one of these parameters:

- Download Only
• Ping
• Upload Only

The synchronization function throws an exception, such as SQLCode.SQLCODE_SQLSyncInfoInvalid or its equivalent, if you set more than one of these parameters to true.

See also
• “ULSynchronize method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
• “ULSyncParms class [UltraLite .NET]” [UltraLite - .NET Programming]
• “ULConnection.Synchronize method [UltraLite C++]” [UltraLite - C and C++ Programming]
• “SyncParms class [UltraLiteJ]” [UltraLite - Java Programming]
• “Stream Type synchronization parameter” on page 98
• “User Name synchronization parameter” on page 103
• “Version synchronization parameter” on page 104
• “Download Only synchronization parameter” on page 88
• “Ping synchronization parameter” on page 93
• “Upload Only synchronization parameter” on page 101

Additional Parameters synchronization parameter

This synchronization parameter allows an application to supply additional parameters that cannot be readily specified using any other predefined parameters. Some parameters that are seldom used are specified in this parameter field.

The additional parameters are supplied as a string of keyword=value settings, separated with a semicolon.

Syntax

The syntax varies depending on the API you use. It is not available in UltraLiteJ.

Allowed values

The following properties can be specified as part of the additional parameters setting:

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowDownloadDupRows</td>
<td>Prevents errors from being raised when a synchronization encounters downloaded rows with duplicate primary keys.</td>
</tr>
<tr>
<td></td>
<td>Set this property to 0 to raise errors and roll back the download; otherwise, set to 1 to raise warnings and continue the download.</td>
</tr>
<tr>
<td></td>
<td>This property is only available in UltraLite C/C++.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckpointStore</td>
<td>Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process.</td>
</tr>
<tr>
<td></td>
<td>Set this property to 1 to enable this feature, which is beneficial for large downloads with many updates but slows down synchronization; otherwise, set to 0, which is the default.</td>
</tr>
<tr>
<td>DisableConcurrency</td>
<td>Disallows database access from other threads during synchronization during the upload phase.</td>
</tr>
<tr>
<td></td>
<td>Set this property to 0 to allow concurrent database access; otherwise, set to 1. By default, this property is set to 0.</td>
</tr>
<tr>
<td>TableOrder</td>
<td>Sets the table order required for priority synchronization if the UltraLite default table ordering is not suitable for your deployment.</td>
</tr>
<tr>
<td></td>
<td>Set this property to a list of table names, arranged in the desired order for upload. For UltraLite, use a comma delimited list; for ulsync, use a semicolon delimited list. By default, the order is based on foreign key relationships. Typically, the default is acceptable when the foreign keys on your consolidated database match the UltraLite remote database and there are no foreign key cycles.</td>
</tr>
<tr>
<td></td>
<td>Quote tables names with either single or double quotes. For example, &quot;Customer,Sales&quot; and 'Customer,Sales' are both supported in UltraLite.</td>
</tr>
<tr>
<td></td>
<td>If you include tables that are not included in the synchronization, they are ignored. Any tables that you do not list are appropriately sorted based on the foreign keys defined in the remote database.</td>
</tr>
<tr>
<td></td>
<td>The order of tables on the download is the same as those you define for upload.</td>
</tr>
<tr>
<td></td>
<td>You only need to explicitly set the table order if your UltraLite tables:</td>
</tr>
<tr>
<td></td>
<td>● Are part of foreign key cycles. You must then list all tables that are part of a cycle.</td>
</tr>
<tr>
<td></td>
<td>● Have different foreign key relationships in the consolidated database.</td>
</tr>
</tbody>
</table>

**See also**

- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncParms.AdditionalParms property [UltraLite.NET]” [UltraLite -.NET Programming]
Example

UltraLite for C/C++ applications can set additional parameters as follows:

```c
ul_sync_info info;
// ...
info.additionalParms = UL_TEXT(
    "AllowDownloadDupRows=1;
    CheckpointStore=1;
    DisableConcurrency=1;
    TableOrder=Customer,Sales"
);
```

Authentication Parameters synchronization parameter

Supplies parameters to authentication parameters in MobiLink events.

Syntax

The syntax varies depending on the API you use.

Remarks

Parameters may be a user name and password, for example.

If you use this parameter, you must also supply the number of parameters.

Allowed values

An array of strings. Null is not allowed as a value for any of the strings, but you can supply an empty string.

See also

- “Number of Authentication Parameters parameter” on page 91
- “Authentication parameters” [MobiLink - Server Administration]
- “authenticate_parameters connection event” [MobiLink - Server Administration]
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “SyncParms.setAuthenticationParms method [UltraLiteJ]” [UltraLite - Java Programming]

Example

UltraLite for C/C++ applications can set the parameters as follows:

```c
ul_char * Params[3] = {UL_TEXT("parm1"),
                        UL_TEXT("parm2"),
                        UL_TEXT("parm3")};
// ...
info.num_auth_parms = 3;
info.auth_parms = Params;
```
Authentication Status synchronization parameter

This field is set by a synchronization to report the status of MobiLink user authentication. The MobiLink server provides this information to the client.

Syntax

The syntax varies depending on the API you use.

Allowed values

The allowed values are held in an interface-specific enumeration. For example, for C/C++ applications the enumeration is as follows.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UL_AUTH_STATUS_UNKNOWN</td>
<td>0</td>
<td>Authorization status is unknown, possibly because the connection has not yet synchronized.</td>
</tr>
<tr>
<td>UL_AUTH_STATUS_VALID</td>
<td>1</td>
<td>User ID and password were valid at the time of synchronization.</td>
</tr>
<tr>
<td>UL_AUTH_STATUS_VALID_BUT_EXPIRES_SOON</td>
<td>2</td>
<td>User ID and password were valid at the time of synchronization but will expire soon.</td>
</tr>
<tr>
<td>UL_AUTH_STATUS_EXPIRED</td>
<td>3</td>
<td>Authorization failed: user ID or password have expired.</td>
</tr>
<tr>
<td>UL_AUTH_STATUS_INVALID</td>
<td>4</td>
<td>Authorization failed: bad user ID or password.</td>
</tr>
<tr>
<td>UL_AUTH_STATUS_IN_USE</td>
<td>5</td>
<td>Authorization failed: user ID is already in use.</td>
</tr>
</tbody>
</table>

Remarks

If a custom authenticate_user synchronization script at the consolidated database returns a different value, the value is interpreted according to the rules given in an authenticate_user connection event.

If you are implementing a custom authentication scheme, the authenticate_user or authenticate_user_hashed synchronization script must return one of the allowed values of this parameter.

The parameter is set by the MobiLink server, and so is read-only.
See also

- “authenticate_user connection event” [MobiLink - Server Administration]
- “MobiLink users” [MobiLink - Client Administration]
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncResult.AuthStatus property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncResult.getAuthStatus method [UltraLiteJ]” [UltraLite - Java Programming]

Example

UltraLite for C/C++ applications can access the parameter as follows:

```c
ul_sync_info info;
// ...
returncode = info.auth_status;
```

Authentication Value synchronization parameter

This field is set by a synchronization to report results of a custom MobiLink user authentication script. The MobiLink server provides this information to the client.

Syntax

The syntax varies depending on the API you use. It is not available in UltraLiteJ.

Remarks

The values set by the default MobiLink user authentication mechanism are described in the authenticate_user connection event and Authentication Status synchronization parameter.

The parameter is set by the MobiLink server, and so is read-only.

See also

- “authenticate_user connection event” [MobiLink - Server Administration]
- “authenticate_user_hashed connection event” [MobiLink - Server Administration]
- “Authentication Status synchronization parameter” on page 86
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncResult.AuthValue property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncResult.getAuthValue method [UltraLiteJ]” [UltraLite - Java Programming]

Example

UltraLite for C/C++ applications can access the parameter as follows:

```c
ul_sync_info info;
// ...
returncode = info.auth_value;
```
Download Only synchronization parameter

Prevents changes from being uploaded from the UltraLite database during this synchronization.

Syntax

The syntax varies depending on the API you use.

Default

False

Allowed values

Boolean

Conflicts with

Ping and Upload Only

Remarks

Data changes are not uploaded when download-only synchronization occurs. However, information about the schema and the value stored in the progress counter is still uploaded. If the downloaded data conflicts with changes on the remote that have not been uploaded, then the synchronization fails and is rolled back.

When using download-only synchronization on UltraLite Java edition databases, you should regularly perform a full synchronization to reduce the amount of transaction log that is scanned when remote databases are synchronized. Otherwise, the synchronizations take an increasingly long time to complete.

See also

- “Upload Only synchronization parameter” on page 101
- “Synchronization state tracking” on page 2
- “UltraLite Synchronization utility (ulsync)” on page 210
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncParms.DownloadOnly property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncParms.setDownloadOnly method [UltraLiteJ]” [UltraLite - Java Programming]

Example

The following example illustrates how to set the DownloadOnly synchronization parameter using the ulsync utility:

```bash
ulsync -c DBF=myuldb.udb
    "MobiLinkUid=remoteA;ScriptVersion=2;DownloadOnly=ON;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```c
ul_sync_info info;
    // ...
    info.download_only = ul_true;
```
Ignored Rows synchronization parameter

This field is set by a synchronization to indicate that rows were ignored by the MobiLink server during synchronization because of absent scripts.

Syntax

The syntax varies depending on the API you use.

Allowed values

Boolean

Remarks

The parameter is read-only.

See also

- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncResult.IgnoredRows property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncResult.getIgnoredRows method [UltraLiteJ]” [UltraLite - Java Programming]

Example

UltraLite for C/C++ applications can access the parameter as follows:

```c
ul_sync_info info;
// ...
res = info.ignored_rows;
```

Keep Partial Download synchronization parameter

Controls whether UltraLite holds on to the partial download rather than rolling back the changes, when a download fails because of a communications error during synchronization.

Syntax

The syntax varies depending on the API you use. It is not available in UltraLiteJ.

Default

False, which indicates that UltraLite rolls back all changes after a failed download.

Allowed values

Boolean
New Password synchronization parameter

Sets a new MobiLink password associated with the user name.

Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync.

Allowed values

String

Remarks

The parameter is optional.

See also

● “MobiLink users” [MobiLink - Client Administration]
● “UltraLite Synchronization utility (ulsync)” on page 210
● “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
● “ULSyncParms.NewPassword property [UltraLite.NET]” [UltraLite - .NET Programming]
● “SyncParms.setPassword method [UltraLiteJ]” [UltraLite - Java Programming]

Example

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb
  "MobiLinkUid=remoteA;ScriptVersion=2;NewMobiLinkPwd=mynewpassword;Stream=http
  "
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
  // ...
  info.new_password = UL_TEXT( "mlnewpass" );
```
Number of Authentication Parameters parameter

Supplies the number of authentication parameters being passed to authentication parameters in MobiLink events.

Syntax

The syntax varies depending on the API you use. Not required for UltraLiteJ.

Default

No parameters passed to a custom authentication script.

Remarks

The parameter is used together with Authentication Parameters to supply information to custom authentication scripts.

See also

- “Authentication Parameters synchronization parameter” on page 85
- “authenticate_parameters connection event” [MobiLink - Server Administration]
- “Authentication parameters” [MobiLink - Server Administration]
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]

Example

UltraLite for C/C++ applications can set the parameter as follows:

```c
ul_sync_info info;
// ...
info.num_auth_parms = 3;
```

Observer synchronization parameter

Specifies a pointer to a callback function or event handler that monitors synchronization. The signature of the callback function that you need to implement to use is of the type `ul_sync_observer_fn`:

```c
typedef void(UL_CALLBACK_FN *ul_sync_observer_fn)( ul_sync_status * status );
```

Syntax

The syntax varies depending on the API you use.

See also

- “User Data synchronization parameter” on page 102
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncProgressListener interface [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncParms.setSyncObserver method [UltraLiteJ]” [UltraLite - Java Programming]
Example

UltraLite for C/C++ applications can set the parameter as follows:

```c
ul_sync_info info;
// ...
info.observer=callfunction;
```

Partial Download Retained synchronization parameter

This field is set by a synchronization to indicate whether UltraLite applied those changes that were downloaded rather than rolling back the changes when a download fails because of a communications error during synchronization.

Syntax

The syntax varies depending on the API you use. Not supported by UltraLite Java edition databases.

Allowed values

Boolean

Remarks

The parameter is set during synchronization if a download error occurs and a partial download was retained.

Partial downloads are retained only if Keep Partial Download is set to true.

See also

- “Keep Partial Download synchronization parameter” on page 89
- “Resumption of failed downloads” [MobiLink - Server Administration]
- “Resume Partial Download synchronization parameter” on page 95
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncResult.PartialDownloadRetained property [UltraLite.NET]” [UltraLite - .NET Programming]

Example

Access the parameter as follows:

```c
ul_sync_info info;
// ...
returncode=info.partial_download_retained;
```

Password synchronization parameter

Specifies the MobiLink password associated with the user name.

Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync.
Allowed values
String

Remarks
The parameter is optional.

This MobiLink user name and password are different than any database user ID and password, and serve to only identify and authenticate the application to the MobiLink server.

If the MobiLink client already has a password, use the New Password parameter to change it.

See also
- “User Name synchronization parameter” on page 103
- “New Password synchronization parameter” on page 90
- “MobiLink users” [MobiLink - Client Administration]
- “UltraLite Synchronization utility (ulsync)” on page 210
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncParms>Password property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncParms.setPassword method [UltraLiteJ]” [UltraLite - Java Programming]

Example
ulsync can set this parameter as an extended synchronization parameter as follows:

    ulsync -c DBF=myuldb.udb
    "MobiLinkUid=remoteA;ScriptVersion=2;MobiLinkPwd=mypassword;Stream=http"

UltraLite for C/C++ applications can set the parameter as follows:

    ul_sync_info info;
    // ...
    info.password = UL_TEXT( "mypassword" );

Ping synchronization parameter
Confirms communications between the UltraLite client and the MobiLink server. When this parameter is set to true, no synchronization takes place.

Syntax
The syntax varies depending on the API you use. You can also set this parameter with ulsync.

Default
False

Allowed values
Boolean
Remarks

When the MobiLink server receives a ping request, it connects to the consolidated database, authenticates
the user, and then sends the authenticating user status and value back to the client.

If the ping succeeds, the MobiLink server issues an information message. If the ping does not succeed, it
issues an error message.

If the MobiLink user ID cannot be found in the ml_user system table and the MobiLink server is running
with the command line option -zu+, the MobiLink server adds the user to ml_user.

The MobiLink server may execute the following scripts, if they exist, for a ping request:

- begin_connection
- authenticate_user
- authenticate_user_hashed
- authenticate_parameters
- end_connection

See also

- “-pi dbmlsync option” [MobiLink - Client Administration]
- “UltraLite Synchronization utility (ulsync)” on page 210
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++
  Programming]
- “ULSyncParms.PingOnly property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncParms.setPingOnly method [UltraLiteJ]” [UltraLite - Java Programming]

Example

UlSync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb
   "MobiLinkUid=remoteA;ScriptVersion=2;Ping=True;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
// ...
info.ping = ul_true;
```

Publications synchronization parameter

Specifies the publications to be synchronized.

Syntax

The syntax varies depending on the API you use. You can also use this parameter with ulsync.

Default

Synchronize all publications.
Remarks

When synchronizing in C/C++, set the publications synchronization parameter to a **publication list**: a comma-separated list of publication names.

See also

- “Publishing data in UltraLite” on page 72
- “UltraLite publications” on page 77
- “UltraLite Synchronization utility (ulsync)” on page 210
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULConnection.SYNC_ALL_DB field [UltraLite.NET]” [UltraLite - .NET Programming]
- “ULConnection.SYNC_ALL_PUBS field [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncParms.setPublications method [UltraLiteJ]” [UltraLite - Java Programming]

Example

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsnc -c DBF=myuldb.udb
    "MobiLinkUid=remoteA;ScriptVersion=2;Publications=UL_PUB_MYPUB1,UL_PUB_MYPUB2;
    Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
// ...
info.publications = UL_TEXT( "Pubs1,Pubs3" );
```

Resume Partial Download synchronization parameter

Resumes a failed download.

Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync. Not supported by UltraLite Java edition databases.

Default

False

Allowed values

Boolean

Remarks

The synchronization does not upload changes; it only downloads those changes that were to be downloaded in the failed download.
See also

- “Resumption of failed downloads” [MobiLink - Server Administration]
- “Keep Partial Download synchronization parameter” on page 89
- “Partial Download Retained synchronization parameter” on page 92
- “UltraLite Synchronization utility (ulsync)” on page 210
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncParms.ResumePartialDownload property [UltraLite.NET]” [UltraLite - .NET Programming]

Example

UltraLite for C/C++ applications can set the parameter as follows:

```c
ul_sync_info info;
/* ...*/
info.resume_partial_download = ul_true;
```

Send Download Acknowledgement synchronization parameter

Instructs the MobiLink server that the client will provide a download acknowledgement.

Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync.

Default

False

Allowed values

Boolean

See also

- “UltraLite Synchronization utility (ulsync)” on page 210
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncParms.SendDownloadAck property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncParms.setAcknowledgeDownload method [UltraLiteJ]” [UltraLite - Java Programming]

Example

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb
"MobiLinkUid=remoteA;ScriptVersion=2;SendDownloadACK=true;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```c
ul_sync_info info;
/* ...*/
info.send_download_ack = ul_true;
```
Stream Error synchronization parameter

Provides a structure to hold communications error reporting information.

Syntax

The syntax varies depending on the API you use.

Applies to

This parameter applies only to C/C++ interfaces.

Allowed values

The parameter has no default value, and must be explicitly set using one of the supported fields. The ul_stream_error fields are as follows:

- **stream_error_code**  
  For the error code suffixes, see %SQLANY16%\SDK\Include\sserror.h.

- **system_error_code**  
  A system-specific error code. For more information about the error code, you must look at your platform documentation. For Windows platforms, this is the Microsoft Developer Network documentation.

The following are common system errors on Windows:

- **10048 (WSAADDRINUSE)**  
  Address already in use.

- **10053 (WSAECNABORTED)**  
  Software caused connection abort.

- **10054 (WSAECNRESET)**  
  The other side of the communication closed the socket.

- **10060 (WSAETIMEDOUT)**  
  Connection timed out.

- **10061 (WSAECNREFUSED)**  

- **error_string**  
  A string with additional information, if available, for the stream_error_code. The string may or may not be empty. A non-empty error_string value provides information in addition to the stream_error_code value. For example, for a write error (error code 9) the error string is a number showing how many bytes it was trying to write.

Remarks

UltraLite applications other than the UltraLite C++ Component receive communications error information as part of the Sync Result parameter.

The stream_error field is a structure of type ul_stream_error.

```c
typedef struct {
    ss_error_code stream_error_code;
    asa_uint16 alignment;
    asa_int32 system_error_code;
    char error_string[UL_STREAM_ERROR_STRING_SIZE];
} ul_stream_error, * p_ul_stream_error;
```
The structure is defined in \%SQLANY16\%SDK\Include\error.h.

Check for a SQLE_MOBILINK_COMMUNICATIONS_ERROR:

```c
ULConnection * conn;
ul_sync_info info;
...
conn->InitSynchInfo( &info );
if( !conn->Synchronize( &info ) ) {
    ULError const * error = conn->GetLastError();
    char buf[256];
    if( error->GetSQLCode() == SQLE_MOBILINK_COMMUNICATIONS_ERROR ) {
        error->GetString( buf, sizeof(buf) );
        printf( "%s\n", buf );
        // more handling for communication error
    }
}
```

See also

- “MobiLink communication error messages” [Error Messages]
- “Sync Result synchronization parameter” on page 100
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]

### Stream Type synchronization parameter

Sets the MobiLink network protocol to use for synchronization.

#### Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync.

#### Remarks

This parameter is required. It has no default value.

Most network protocols require protocol options to identify the MobiLink server address and other behavior. These options are supplied in the Stream Parameters parameter.

When the network protocol requires an option, pass that option using the Stream Parameters parameter; otherwise, set the Stream Parameters parameter to null.

The following stream types are available, but not all are available on all target platforms:

<table>
<thead>
<tr>
<th>Network protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>Synchronize over HTTP.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Synchronize over HTTPS.</td>
</tr>
<tr>
<td></td>
<td>The HTTPS protocol uses TLS as its underlying security layer. It operates over TCP/IP.</td>
</tr>
<tr>
<td>Network protocol</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Synchronize over TCP/IP. This protocol is not supported by UltraLite Java edition databases.</td>
</tr>
<tr>
<td>TLS</td>
<td>Synchronize over TCP/IP with transport-layer security (TLS). TLS secures client/server communications using digital certificates and public-key cryptography.</td>
</tr>
</tbody>
</table>

For a list of supported platforms, see [http://www.sybase.com/detail?id=1061806](http://www.sybase.com/detail?id=1061806).

**See also**

- “Stream Parameters synchronization parameter” on page 99
- “Certificate Creation utility (createcert)” [SQL Anywhere Server - Database Administration]
- “Certificate Viewer utility (viewcert)” [SQL Anywhere Server - Database Administration]
- “Transport-layer security” [SQL Anywhere Server - Database Administration]
- “UltraLite Synchronization utility (ulsync)” on page 210
- “UltraLite network protocol options” on page 104
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncParms.Stream property [UltraLite.NET]” [UltraLite - .NET Programming]

**Example**

For UltraLite for C/C++ applications, set the parameter as follows:

```c
Connection conn;
ul_sync_info info;
... conn.InitSynchInfo( &info );
info.stream = "http";
```

**Stream Parameters synchronization parameter**

Sets options to configure the network protocol.

**Syntax**

The syntax varies depending on the API you use. You can also set this parameter with ulsync.

**Default**

Null

**Allowed values**

String

**Remarks**

This parameter is optional. It accepts a semicolon separated list of network protocol options. Each option is of the form keyword=value, where the allowed sets of keywords depends on the network protocol.
See also

- “UltraLite Synchronization utility (ulsync)” on page 210
- “UltraLite network protocol options” on page 104
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncParms.StreamParms property [UltraLite.NET]” [UltraLite - .NET Programming]
- “StreamHTTPParms interface [UltraLiteJ]” [UltraLite - Java Programming]

Example

UltraLite for C/C++ applications can set the parameter as follows:

```c
ul_sync_info info;
// ...
info.stream_parms= UL_TEXT( "host=myserver;port=2439" );
```

**Sync Result synchronization parameter**

Reports the status of a synchronization.

**Syntax**

The syntax varies depending on the API you use.

**Remarks**

The parameter is set by UltraLite, and is read-only.

The C/C++ interface receives this information in separate parameters as part of a ul_sync_info struct. Otherwise, this information is defined as a compound parameter containing a variety of information in separate fields:

- **Authentication Status**  Reports success or failure of authentication.
- **Ignored Rows**  Reports the number of ignored rows.
- **Stream Error information**  The Stream Error information includes a Stream Error Code, Stream Error Context, Stream Error ID, and Stream Error System.
- **Upload OK**  Reports the success or failure of the upload phase.
See also

- “Authentication Status synchronization parameter” on page 86
- “Ignored Rows synchronization parameter” on page 89
- “Stream Error synchronization parameter” on page 97
- “Upload OK synchronization parameter” on page 101
- “ULSyncParms class [UltraLite.NET]” [UltraLite - .NET Programming]
- “ul_sync_result structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULGetSyncResult method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “SyncResult class [UltraLiteJ]” [UltraLite - Java Programming]

Upload OK synchronization parameter

This field is set by a synchronization to report the status of data uploaded to the MobiLink server.

Syntax

The syntax varies depending on the API you use.

Remarks

The parameter is set by UltraLite, and so is read-only.

After synchronization, the parameter holds true if the upload was successful, and false otherwise. You can check this parameter if there was a synchronization error, to know whether data was successfully uploaded before the error occurred.

See also

- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncResult.UploadOK property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncResult.isUploadOK method [UltraLiteJ]” [UltraLite - Java Programming]

Example

UltraLite for C/C++ applications can access the parameter as follows:

```c
ul_sync_info info;
// ...
returncode = info.upload_ok;
```

Upload Only synchronization parameter

Indicates that there should be no downloads in the current synchronization, which can save communication time, especially over slow communication links.

Syntax

The syntax varies depending on the API you use. You can also set this parameter with ulsync.
Default
False

Allowed values
Boolean

Conflicts with
Download Only, Ping, and Resume Partial Download

Remarks
When set to true, the client waits for the upload acknowledgement from the MobiLink server, after which it terminates the synchronization session successfully.

See also
- “UltraLite client synchronization design” on page 68
- “Download Only synchronization parameter” on page 88
- “UltraLite Synchronization utility (ulsync)” on page 210
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncParms.UploadOnly property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncParms.isUploadOnly method [UltraLiteJ]” [UltraLite - Java Programming]

Example
ulsync can set this parameter as an extended synchronization parameter as follows:

ulsync -c DBF=myuldb.udb
"MobiLinkUid=remoteA;ScriptVersion=2;UploadOnly=True;Stream=http"

UltraLite for C/C++ applications can set the parameter as follows:

ul_sync_info info;
// ...
info.upload_only = ul_true;

User Data synchronization parameter
Makes application-specific information available to the synchronization observer.

Applies to
C/C++ applications only. Other components, such as UltraLite.NET, do not require a separate parameter to handle user data and so have no User Data parameter.

Syntax
The syntax varies depending on the API you use.
Remarks
When implementing the synchronization observer callback function or event handler, you can make application-specific information available by providing information using the User Data parameter.

See also
- “Observer synchronization parameter” on page 91
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]

User Name synchronization parameter
Required. A string that the MobiLink server uses for authentication purposes.

Syntax
The syntax varies depending on the API you use. You can also set this parameter with ulsync.

Remarks
This parameter is required. Empty strings and NULL strings are universally rejected.

The parameter has no default value, and must be explicitly set.

The user name does not have to be unique when a remote ID is used.

This MobiLink user name and password are separate from any database user ID and password, and serves only to identify and authenticate the application to the MobiLink server.

For a user to be part of a synchronization system, you must register the user name with the MobiLink server. The user name is stored in the name column of the ml_user MobiLink system table in the consolidated database.

See also
- “Remote IDs” [MobiLink - Client Administration]
- “Password synchronization parameter” on page 92
- “MobiLink users” [MobiLink - Client Administration]
- “UltraLite user authentication” [MobiLink - Client Administration]
- “UltraLite Synchronization utility (ulsync)” on page 210
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “ULSyncParms.UserName property [UltraLite.NET]” [UltraLite - .NET Programming]
- “SyncParms.setUserName method [UltraLiteJ]” [UltraLite - Java Programming]

Example
ulsync can set this parameter as an extended synchronization parameter as follows:

    ulsync -c DBF=myuldb.udb "MobiLinkUid=remoteA;ScriptVersion=2;Stream=http"

UltraLite for C/C++ applications can set the parameter as follows:
ul_sync_info info;
   // ...
   info.user_name = UL_TEXT( "remoteA" );

### Version synchronization parameter

Defines the consolidated database version.

**Syntax**

The syntax varies depending on the API you use. You can also set this parameter with ulsync.

**Allowed values**

String

**Remarks**

This parameter is required. Empty strings and NULL strings are universally rejected.

Each synchronization script in the consolidated database is marked with a version string. For example, there may be two different download_cursor scripts, identified by different version strings.

**See also**

- “Script versions” [MobiLink - Server Administration]
- “UltraLite Synchronization utility (ulsync)” on page 210
- “ul_sync_info structure [UltraLite C and Embedded SQL datatypes]” [UltraLite - C and C++ Programming]
- “SyncParms.setVersion method [UltraLiteJ]” [UltraLite - Java Programming]

**Example**

ulsync can set this parameter as an extended synchronization parameter as follows:

```
ulsync -c DBF=myuldb.udb "MobiLinkUid=remoteA;ScriptVersion=2;Stream=http"
```

UltraLite for C/C++ applications can set the parameter as follows:

```
ul_sync_info info;
   // ...
   info.version = UL_TEXT( "default" );
```

### UltraLite network protocol options

You must set the network protocol in your application. Each UltraLite database that synchronizes with a MobiLink server does so over a network protocol. Available network protocols include TCP/IP, HTTP, HTTPS, and TLS. Support is also provided for ActiveSync notification on Windows Mobile.

For the network protocol you set, you can choose from a set of corresponding protocol options to ensure that the UltraLite application can locate and properly communicate with the MobiLink server. The
MobiLink client network protocol options provide information such as addressing information (host and port) and protocol-specific information.

See also

- “MobiLink client network protocol options” [MobiLink - Client Administration]
- “Configuring UltraLite clients to use transport-layer security” [SQL Anywhere Server - Database Administration]
- “MobiLink client network protocol options” [MobiLink - Client Administration]
- “Stream Parameters synchronization parameter” on page 99
- -x option in “UltraLite Synchronization utility (ulsync)” on page 210

### Synchronization stream options

You can provide the information needed to locate the MobiLink server in your application by setting the Stream Parameters parameter.

See also

- “Stream Parameters synchronization parameter” on page 99
- “ULSyncParms.StreamParms property [UltraLite.NET]” [UltraLite - .NET Programming]
UltraLite deployment

In the majority of cases, development occurs on a Windows desktop or Mac OS X with the final release target for UltraLite being the mobile device. However, depending on your deployment environment, you can use various deployment mechanisms to install UltraLite.

UltraLite application projects may evolve with different iterations of the same UltraLite database: a development database, a test database, and a deployed production database. During the lifetime of a deployed database application, changes and improvements are first made in the development database, then propagated to the test database, before finally being distributed to the production database.

The modules you need to use for your UltraLite application depend on the platform you are targeting, the interface you are using, and the functionality you want to use.

See also

- “UltraLite data management components for Windows Mobile” on page 16
- “How to build and deploy UltraLite C++ applications” [UltraLite - C and C++ Programming]
- “UltraLite.NET application development” [UltraLite - .NET Programming]
- “How to build and deploy UltraLiteJ applications” [UltraLite - Java Programming]

UltraLite application build and deployment specifications

[ This topic has been updated for build 1823. ]

This section contains tables that show the build and deployment requirements for UltraLite applications.

Note

There may be versions of the UltraLite engine located in directories that contains the _dev suffix, such as the x86_dev directory. These versions contain development-time logging functionality that can be used to diagnose problems on platforms for debugging purposes. For production systems, use a version of the engine that is not in a _dev directory.

Build and deployment requirements for UltraLite applications and encryption

The following table describes the minimum requirements to build and deploy an UltraLite application for all supported platforms and devices, including the requirements for UltraLite database encryption.
<table>
<thead>
<tr>
<th>Platform or device</th>
<th>Minimum requirements</th>
<th>AES encryption requirements</th>
<th>FIPS 140-2 AES encryption requirements</th>
</tr>
</thead>
</table>
| Windows Mobile and desktop (UltraLite C/C++ using static linkage) | Link against:  
  - `ulrt.lib`\(^1\)  
  - `ulbase.lib`\(^1\) | Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.  
  Call the EnableAesDBEncryption method. | Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.  
  Set the creation parameter `fips=yes` when creating the database.  
  Call the EnableAesFipsDBEncryption method.  
  Deploy:  
  - `ulfips16.dll`\(^2\)  
  - `libeay32.dll`\(^2\)  
  - `msvcr90.dll`\(^2\) |
| Windows Mobile and desktop (UltraLite C/C++ using dynamic linkage) | Link against:  
  - `ulimp.lib`\(^1,10\)  
  - `ulbase.lib`\(^1\)  
  Deploy:  
  - `ulrt16.d11`\(^1\) | Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.  
  Call the EnableAesDBEncryption method. | Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.  
  Set the creation parameter `fips=yes` when creating the database.  
  Call the EnableAesFipsDBEncryption method.  
  Deploy:  
  - `ulfips16.dll`\(^2\)  
  - `libeay32.dll`\(^2\)  
  - `msvcr90.dll`\(^2\) |
<table>
<thead>
<tr>
<th>Platform or device</th>
<th>Minimum requirements</th>
<th>AES encryption requirements</th>
<th>FIPS 140-2 AES encryption requirements</th>
</tr>
</thead>
</table>
| Windows Mobile and desktop (UltraLite C/C++ with the UltraLite engine) | Link against:  
  - ulrtc.lib\(^1\)  
  - ulbase.lib\(^1\)  
Deploy:  
  - uleng16.exe\(^2\) | Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database. | Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.  
  Set the creation parameter \texttt{fips=yes} when creating the database.  
  Deploy:  
    - ulfips16.dll\(^2\)  
    - libeay32.dll\(^2\)  
    - msvcr90.dll \(^2\) |
| Windows Mobile and desktop (UltraLite.NET)             | Add references to:  
  - iAnywhere.Data.UltraLite  
  - iAnywhere.Data.UltraLite.resources  
Deploy:  
  - iAnywhere.Data.UltraLite.dll\(^7\)  
  - iAnywhere.Data.UltraLite.resources.dll\(^8\)  
  - ulnet16.dll\(^6\) | Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database. | Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.  
  Set the creation parameter \texttt{fips=yes} when creating the database.  
  Deploy:  
    - ulfips16.dll\(^2\)  
    - libeay32.dll\(^2\)  
    - msvcr90.dll \(^2\) |
<table>
<thead>
<tr>
<th>Platform or device</th>
<th>Minimum requirements</th>
<th>AES encryption requirements</th>
<th>FIPS 140-2 AES encryption requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Mobile and desktop (UltraLite.NET with the UltraLite engine)</td>
<td>Add references to:</td>
<td>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</td>
<td>Set the DBKEY creation parameter to the encryption key when creating and connecting to the database.</td>
</tr>
<tr>
<td></td>
<td>- iAnywhere.Data.UltraLite</td>
<td></td>
<td>Set the creation parameter <strong>fips=yes</strong> when creating the database.</td>
</tr>
<tr>
<td></td>
<td>- iAnywhere.Data.UltraLite.resources</td>
<td></td>
<td>Deploy:</td>
</tr>
<tr>
<td></td>
<td>Deploy:</td>
<td></td>
<td>- ulfips16.dll²</td>
</tr>
<tr>
<td></td>
<td>- iAnywhere.Data.UltraLite.dll⁷</td>
<td></td>
<td>- libeay32.dll²</td>
</tr>
<tr>
<td></td>
<td>- iAnywhere.Data.UltraLite.resources.dll⁸</td>
<td></td>
<td>- msvcr90.dll²</td>
</tr>
<tr>
<td></td>
<td>- ulnetclient16.dll⁶</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- uleng16.exe²</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mac OS X and iOS (UltraLite C/C++)</td>
<td>Add to your Xcode project:</td>
<td>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</td>
<td>Not applicable</td>
</tr>
<tr>
<td></td>
<td>- libulrt.a⁹</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- libulbase.a⁹ (Mac OS X only)</td>
<td>Call the EnableAesDBEncryption method.</td>
<td></td>
</tr>
<tr>
<td>Linux (UltraLite C/C++)</td>
<td>Link against:</td>
<td>Use the DBKEY creation parameter to set the encryption key when creating or connecting to the database.</td>
<td>Not applicable</td>
</tr>
<tr>
<td></td>
<td>- libulrt.a³</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- libulbase.a³</td>
<td>Call the EnableAesDBEncryption method.</td>
<td></td>
</tr>
<tr>
<td>Android (UltraLiteJ)</td>
<td>Add to your Android project:</td>
<td>Use the DBKEY creation parameter or the setEncryptionKey method to set the encryption key when creating or connecting to the database.</td>
<td>Not applicable</td>
</tr>
<tr>
<td></td>
<td>- UltraLiteJNI16.jar⁵</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- libultralitej16.so⁴</td>
<td>Call the EnableAesDBEncryption method.</td>
<td></td>
</tr>
</tbody>
</table>

---

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
<table>
<thead>
<tr>
<th>Platform or device</th>
<th>Minimum requirements</th>
<th>AES encryption requirements</th>
<th>FIPS 140-2 AES encryption requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackBerry (UltraLiteJ)</td>
<td>Deploy:</td>
<td>Use the setEncryptionKey method to set the encryption key when creating or connecting to the database.</td>
<td>Not applicable</td>
</tr>
<tr>
<td></td>
<td>● UltraLiteJ16.cod11</td>
<td>Call the EnableAesDBEncryption method.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● UltraLiteJ16.jad112</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 For Windows Mobile, this file is located in `%SQLANY16%\UltraLite\CE\Arm.50\Lib`. For Windows, it is located in `%SQLANY16%\UltraLite\Windows\x64\Lib\VS9` or `%SQLANY16%\UltraLite\Windows\x86\Lib\VS9`.

2 FIPS is not supported on Windows Mobile. On Windows, these files are located in `%SQLANY16%\UltraLite\Windows\x64` or `%SQLANY16%\UltraLite\Windows\x86`.

3 This file is located in `/opt/sqlanywhere16/ultralite/linux/x64/lib`.

4 This file is located in `%SQLANY16%\UltraLite\UltraLite\Android\ARM`.

5 This file is located in `%SQLANY16%\UltraLite\UltraLite\Android`.

6 For Windows Mobile, this file is located in `%SQLANY16%\UltraLite\UltraLite.NET\CE\Arm.50`. For Windows, it is located in `%SQLANY16%\UltraLite\UltraLite.NET\x64` or `%SQLANY16%\UltraLite\UltraLite.NET\win32`.

7 This file is located in `%SQLANY16%\UltraLite\UltraLite.NET\Assembly\V2`.

8 This file is located in `%SQLANY16%\UltraLite\UltraLite.NET\Assembly\V2\en`.

9 For Mac OS X, this file is located in `/Applications/SQLAnywhere16/System/ultralite/macosx/x86_64`. For iOS, UltraLite runtimes must be built after installation. Follow the instructions provided in `install-dir/ultralite/iphone/readme.txt`.

10 When linking against this library, define the UL_USE_DLL preprocessor macro when compiling. For example, specify the following:

   `-DUL_USE_DLL`

11 This file is located in `%SQLANY16%\UltraLite\UltraLite\BlackBerry4.2`.

12 Required for over-the-air (OTA) deployment only. Alternatively, you can create your own .jad file that deploys UltraLiteJ with your application.

**Note**
To use UltraLite database obfuscation on any platform or device, you must specify the creation parameter `obfuscate=1` when creating the database.
Additional build and deployment requirements for synchronization and compression

The following table describes the stream, protocol option, and code requirements for building and deploying an UltraLite application that uses synchronization.

Note
The HTTPS stream option can be enabled in the UltraLiteJ API by passing the SyncParms.HTTPS_STREAM constant to the Connection.createSyncParms method.

<table>
<thead>
<tr>
<th>Synchronization type</th>
<th>Stream option specification</th>
<th>Protocol option requirements</th>
<th>Method call requirements for UltraLite C and C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP</td>
<td>&quot;tcpip&quot;</td>
<td>None</td>
<td>● EnableTcpipSynchronization</td>
</tr>
<tr>
<td>HTTP</td>
<td>&quot;http&quot;</td>
<td>None</td>
<td>● EnableHttpSynchronization</td>
</tr>
<tr>
<td>RSA TLS</td>
<td>&quot;tls&quot;</td>
<td>None</td>
<td>● EnableTlsSynchronization● EnableRsaSyncEncryption</td>
</tr>
<tr>
<td>RSA HTTPS</td>
<td>&quot;https&quot;</td>
<td>None</td>
<td>● EnableHttpsSynchronization ● EnableRsaSyncEncryption</td>
</tr>
<tr>
<td>RSA FIPS 140-2 TLS</td>
<td>&quot;tls&quot;</td>
<td>fips=yes</td>
<td>● EnableTlsSynchronization ● EnableRsaFipsSyncEncryption</td>
</tr>
<tr>
<td>RSA FIPS 140-2 HTTPS</td>
<td>&quot;https&quot;</td>
<td>fips=yes</td>
<td>● EnableHttpsSynchronization ● EnableRsaFipsSyncEncryption</td>
</tr>
</tbody>
</table>

The following table describes additional protocol option and code requirements for building and deploying an UltraLite application that uses compression or end-to-end encryption:

<table>
<thead>
<tr>
<th>Compression and stream encryption options</th>
<th>Protocol option requirements</th>
<th>Method call requirements for UltraLite C/C++ and UltraLiteJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZLIB compression</td>
<td>● compression=zlib</td>
<td>● C/C++: EnableZlibSyncCompression ● Java: setZlibCompression</td>
</tr>
<tr>
<td>RSA E2EE</td>
<td>● e2ee_public_key=key-file</td>
<td>● C/C++: EnableRsaE2ee ● Java: setE2eePublicKey</td>
</tr>
<tr>
<td>RSA FIPS 140-2 E2EE</td>
<td>● e2ee_public_key=key-file</td>
<td>● C/C++: EnableRsaFipsE2ee ● Java: Not applicable</td>
</tr>
</tbody>
</table>

The following table illustrates additional build and deployment requirements for compression and encrypted synchronization.
Note
There are no additional build and deployment requirements for TCP/IP and HTTP synchronization.

<table>
<thead>
<tr>
<th>Platform or device</th>
<th>ZLIB compression requirements</th>
<th>RSA TLS, RSA HTTPS, and RSA E2EE requirements</th>
<th>RSA FIPS 140-2 TLS, RSA FIPS 140-2 HTTPS, and RSA FIPS 140-2 E2EE requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Mobile and desktop (UltraLite C/C+ + using static linkage)</td>
<td>None</td>
<td>Link against:</td>
<td>Deploy:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● ulrsa.lib¹</td>
<td>● mlcrsafips16.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● libeay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● ssleay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● msvcr90.dll²</td>
</tr>
<tr>
<td>Windows Mobile and desktop (UltraLite C/C+ + using dynamic linkage)</td>
<td>Deploy:</td>
<td>Deploy:</td>
<td>Deploy:</td>
</tr>
<tr>
<td></td>
<td>● mlczlib16.dll²</td>
<td>● mlcrsa16.dll²</td>
<td>● mlcrsafips16.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● libeay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● ssleay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● msvcr90.dll²</td>
</tr>
<tr>
<td>Windows Mobile and desktop (UltraLite C/C+ + with the UltraLite engine)</td>
<td>Deploy:</td>
<td>Deploy:</td>
<td>Deploy:</td>
</tr>
<tr>
<td></td>
<td>● mlczlib16.dll²</td>
<td>● mlcrsa16.dll²</td>
<td>● mlcrsafips16.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● libeay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● ssleay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● msvcr90.dll²</td>
</tr>
<tr>
<td>Windows Mobile and desktop (UltraLite.NET)</td>
<td>Deploy:</td>
<td>Deploy:</td>
<td>Deploy:</td>
</tr>
<tr>
<td></td>
<td>● mlczlib16.dll²</td>
<td>● mlcrsa16.dll²</td>
<td>● mlcrsafips16.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● libeay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● ssleay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● msvcr90.dll²</td>
</tr>
<tr>
<td>Windows Mobile and desktop (UltraLite.NET with the UltraLite engine)</td>
<td>Deploy:</td>
<td>Deploy:</td>
<td>Deploy:</td>
</tr>
<tr>
<td></td>
<td>● mlczlib16.dll²</td>
<td>● mlcrsa16.dll²</td>
<td>● mlcrsafips16.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● libeay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● ssleay32.dll²</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>● msvcr90.dll²</td>
</tr>
<tr>
<td>Mac OS X and iOS (UltraLite C/C++)</td>
<td>None</td>
<td>None</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Platform or device</td>
<td>ZLIB compression requirements</td>
<td>RSA TLS, RSA HTTPS, and RSA E2EE requirements</td>
<td>RSA FIPS 140-2 TLS, RSA FIPS 140-2 HTTPS, and RSA FIPS 140-2 E2EE requirements</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------</td>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Linux (UltraLite C/C++)</td>
<td>None</td>
<td>Link against:</td>
<td>Not applicable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● libulrsa.a&lt;sup&gt;3&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>Android (UltraLiteJ)</td>
<td>None</td>
<td>Deploy:</td>
<td>Not applicable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● libmlcrsa16.so&lt;sup&gt;4&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>BlackBerry (UltraLiteJ)</td>
<td>None</td>
<td>Transfer the DER encoded file using the File-Transfer interface, or store it on an SD card.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

<sup>1</sup> For Windows Mobile, this file is located in `%SQLANY16%\UltraLite\CE\Arm.50\Lib`. For Windows, it is located in `%SQLANY16%\UltraLite\Windows\x64\Lib\VS9` or `%SQLANY16%\UltraLite\Windows\x86\Lib\VS9`.

<sup>2</sup> FIPS is not supported on Windows Mobile. On, Windows, these files are located in `%SQLANY16%\Windows\x64` or `%SQLANY16%\Windows\x86`.

<sup>3</sup> This file is located in `/opt/sqlanywhere16/ultralite/linux/x64/lib`.

<sup>4</sup> This file is located in `%SQLANY16%\UltraLite\UltraLiteJ\Android\ARM`.

See also
- “How to build and deploy UltraLite C++ applications” [UltraLite - C and C++ Programming]
- “How to build and deploy UltraLite.NET applications” [UltraLite - .NET Programming]
- “How to build and deploy UltraLiteJ applications” [UltraLite - Java Programming]
- “MobiLink client network protocol options” [MobiLink - Client Administration]
UltraLite and UltraLite Java edition database deployment techniques

You can use one of the following techniques to get the initial database file onto the device:

- Use the UltraLite API in your application to create the initial database file.
- Create a schema file from a SQL script and use the `ALTER DATABASE SCHEMA FROM FILE` statement.\(^1\)
- Use UltraLite FileTransfer methods to download the initial database file if it does not already exist on the device.
- Bundle the initial database with the application.\(^1\)
- When deploying to a Windows or Windows Mobile device, use central administration to send down the initial UDB file, or send a command to create the initial database file and execute SQL to give it schema.\(^1\)

\(^1\) Not supported for UltraLite Java edition databases.

See also

- “Deploying UltraLite database schema upgrades” on page 115
- “ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]” on page 397
- “Central administration of remote databases” [MobiLink - Server Administration]

Deploying UltraLite database schema upgrades

Perform a schema upgrade.

Prerequisites

The SQL file you use must contain the entire new schema.

Context and remarks

UltraLite database schema upgrades can be deployed using one of the following techniques:

- **Individual DDL statements** For example, in UltraLite C, you can run the following statement to create a new publication:

  ```c
  dbconnection->ExecuteStatement("CREATE PUBLICATION p (table t)");
  ```

- **The ALTER DATABASE SCHEMA FROM FILE statement** This statement can be used to perform schema upgrades when you do not know the DDL statement requirements, or do not want to specify the individual DDL statements.

  Caution
  Do not reset a device during a schema upgrade. If you reset the device during a schema upgrade, data will be lost and the UltraLite database marked as "bad."
UltraLite executes the following steps when you upgrade an UltraLite database schema with the ALTER DATABASE SCHEMA FROM FILE statement:

1. Both the new and existing database schemas are compared to see what differs.
2. The schema of the existing database is altered.
3. Rows that do not fit the new schema are dropped. For example:
   - If you add a uniqueness constraint to a table and there are multiple rows with the same values, all but one row will be dropped.
   - If you try to change a column domain and a conversion error occurs, then that row is dropped. For example, if you have a VARCHAR column and convert it to an INT column and a row has the value ABCD, then that row is dropped.
   - If your new schema has new foreign keys where the foreign row does not have a matching primary row, these rows are dropped.
4. When rows are dropped, a SQLE_ROW_DROPPED_DURING_SCHEMA_UPGRADE (130) warning is raised.

**Task**

1. Create a SQL script of DDL statements to create a completely new schema.

You can keep a master schema on your computer and update the schema as your application changes. Use either the ulinit or ulunload utilities to extract the DDL statements required for your script. By using these utilities with the following options, you ensure that the DDL statements are syntactically correct:

- For an UltraLite database, use the ulunload utility with the -n and -s [schema-file ] options. For example:
  
  ```
  ulunload -c dbf=mydatabase.udb -n -s MySchema.sql
  ```

- For a SQL Anywhere database, use the ulinit utility with the -a and -l [schema-file ] option. For example:
  
  ```
  ulinit -a "dsn=mysqlanywheredatabase" -l MySchema.sql
  ```

If you do not use the ulunload or ulinit utilities, review the script and ensure the following:

- The script declares the entire desired schema with CREATE statements.
- Tables, columns, and publications are not renamed. The RENAME operation is not supported. Renamed tables are processed as a DROP TABLE and CREATE TABLE operation.
- There are no non-DDL statements, including non-DDL statements that may not have the effect you expect.
- Words in the SQL statement are separated by spaces.
2. Deploy the new SQL script file.

3. Ensure that the database is synchronized.

4. Run the new statement on the device. For example:

   ALTER DATABASE SCHEMA FROM FILE 'MySchema.sql'

   **Results**

   The schema is updated.

   **Error notification**

   Because UltraLite error callback is active during the upgrade process, you are notified of errors during the conversion process. For example, SQLE_CONVERSION_ERROR reports all values that could not be converted in its parameters. Errors do not mean the process failed. The final SQL code after the statement returns is a 130 warning in this case. These warnings describe operations of the conversion process and do not stop the upgrade process.

   **See also**

   - “UltraLite database schemas” on page 45
   - “ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]” on page 397
   - “UltraLite Initialize Database utility (ulinit)” on page 198
   - “UltraLite Database Unload utility (ulunload)” on page 216

**UltraLite engine startup**

When using the UltraLite engine to manage data on a Windows or Windows Mobile device, your UltraLite application starts the engine automatically unless the application needs to explicitly provide the directory location of the engine.

When an UltraLite application attempts to start the UltraLite engine, the application searches the following directories:
## Client platform | Directory locations
---|---
Windows desktop | 1. The directory of the application that is starting it  
2. The current working directory  
3. The system path  
4. The SQL Anywhere install directory (either under `bin32` or `bin64`), depending on whether the client is 32-bit or 64-bit
Windows Mobile/CE | 1. `%Windows\`  
2. `\` (the root directory)  
3. `\UltraLiteDB\`
Linux | 1. The directory of the application that is auto-starting it  
2. `%SQLANY16%/bin32`

If the UltraLite engine is stored in a different location, start the engine by specifying the `START` connection parameter.

For example, a connection string to the database or connection code for a Windows Mobile client application might use the following `START` parameter value:

```
"START=\Program Files\MyApp\uleng16.exe"
```

**See also**

- “UltraLite application build and deployment specifications” on page 107
- “UltraLite START connection parameter” on page 175

### Deploying the ActiveSync provider for UltraLite

When you deploy UltraLite for the end user, you must manually install and register the ActiveSync provider on the end user's computer. This requirement ensures that ActiveSync knows when to call a specific instance of a provider for a specific application.

**Prerequisites**

Ensure that the end user has:

- The ActiveSync Manager installed.
- The ActiveSync provider files copied from a development computer to the user's hard drive.
Context and remarks

The UltraLite ActiveSync provider is a software module that lets users gain access to their devices from the desktop. Like other software components, you need to deploy the necessary files to the device to ensure that UltraLite operates with Windows Mobile ActiveSync:

- **mlasinst.exe** Installs the ActiveSync provider and registers it with the ActiveSync Manager. This utility also registers applications with the ActiveSync provider for synchronization.

- **mlasdesk.dll** The DLL that is loaded by the ActiveSync Manager on the desktop. *mlasinst.exe* registers the location of this file with the ActiveSync Manager.

- **mlasdev.dll** The DLL that is loaded by the ActiveSync Manager on the device. *mlasinst.exe* deploys this file to the correct location on the device.

- **dblgen16.dll** The language resource library.

For a list of supported provider platforms, see [http://www.sybase.com/detail?id=1002288](http://www.sybase.com/detail?id=1002288).

Task

1. Run *mlasinst* to install a provider for ActiveSync. You can also use it to register and deploy the UltraLite application to the user's Windows Mobile device—depending on the command-line syntax you use. If your UltraLite application uses multiple files, you must manually copy the required files.

   The following example assumes that both *mlasdesk.dll* and *mlasdev.dll* are in the current directory. The -k and -v options are used. The -p and -x options are command-line options for the application when it is started by ActiveSync.

   ```
   mlasinst "C:\My Files\myULapp.exe" "\Program Files\myULapp.exe"
   "My Application" MYAPP -p -x -v -k
   ```

   If you were to use this utility to deploy a pre-compiled CustDB for the ARM 5.0 processor, the command line would be similar to the following one:

   ```
   mlasinst -v "%SQLANY16\UltraLite\ce\arm.50"
   "%SQLANY16\UltraLite\ce\arm.50\custdb.exe" custdb.exe CustDB CUSTDBDEMO
   ```

   **Note**
   You can also use the ActiveSync to register your UltraLite application at a later time if you choose. See “Registering applications with the ActiveSync Manager” on page 120.

2. Restart your computer so ActiveSync can recognize the new provider.

3. Enable the MobiLink provider.
   a. From the ActiveSync window, click **Options**.
   b. Check **MobiLink Clients** in the list and click **OK** to activate the provider.
   c. To see a list of registered applications, click **Options**, click **MobiLink Clients**, and click **Settings**.
Results

The files are deployed to the device, allowing UltraLite to operate with Windows Mobile ActiveSync.

See also

● “Registering applications with the ActiveSync Manager” on page 120
● “Microsoft ActiveSync Provider Installation utility (mlasinst)” [MobiLink - Client Administration]

Registering applications with the ActiveSync Manager

Applications that use ActiveSync synchronization must be registered with ActiveSync and copied to the device.

Prerequisites

There are no prerequisites for this task.

Context and remarks

You can register your application for use with ActiveSync either by using the ActiveSync Provider Installation utility or using the ActiveSync Manager itself. This section describes how to use the ActiveSync Manager.

Task

1. Launch ActiveSync.

2. From the ActiveSync window, click Options.

3. From the list of information types, click MobiLink Clients and click Settings.

4. In the MobiLink Synchronization window, click New.

5. Enter the following information for your application:
   
   ● Application name A name identifying the application that appears in the ActiveSync user interface.
   
   ● Class name The registered class name for the application.
   
   ● Path The location of the application on the device.
   
   ● Arguments Any command line arguments to be used when ActiveSync starts the application.

6. Click OK to register the application.

Results

The application is registered with ActiveSync.
Next

Copy the application to the device.

See also

- “Assigning class names for applications” [UltraLite - C and C++ Programming]
- “Microsoft ActiveSync Provider Installation utility (mlasinst)” [MobiLink - Client Administration]
Tutorial: Building the UltraLite CustDB sample application

In this tutorial, you learn how to:

- Run the MobiLink server to carry out data synchronization between the consolidated database and the UltraLite remote.
- Use Sybase Central to browse the data in the UltraLite remote.
- Manage UltraLite databases with UltraLite utilities.

Different versions of the application code exist for each supported programming interface and platform. However, this tutorial references the compiled version of the application for Windows desktops only. Each version varies to conform to the conventions of each platform.

See also
- “CustDB sample application overview” on page 12
- “CustDB Scenario” [MobiLink - Getting Started]
- “Users in the CustDB sample” [MobiLink - Getting Started]
- “Tables in the CustDB databases” [MobiLink - Getting Started]

Lesson 1: Building and running the CustDB application

In this lesson, you build and run the CustDB application.

Prerequisites

There are no prerequisites for this task.

Task

1. For non-Windows environments, build the CustDB application.
   a. Open a CustDB project file in the appropriate environment.
   b. Compile the source code.

2. Run the CustDB application.

   For Windows 32-bit environments, run %SQLANY16%\UltraLite\Windows\x86\custdb.exe.

Results

The CustDB application runs.
Lesson 2: Starting the MobiLink server and performing an initial synchronization

In this lesson, you start the MobiLink server and synchronize the CustDB database with the UltraLite database using the CustDB application.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Building and running the CustDB application” on page 123.

Task

1. Click Start » Programs » SQL Anywhere 16 » MobiLink » Synchronization Server Sample. Or, run the following command:

   ```bash
   mlsrv16 -c "DSN=SQL Anywhere 16 CustDB" -vcrs
   ```

   Use mobilink.sh on Mac OS X or Linux.

   The window displays messages about the MobiLink server's status.

2. Click Start » Programs » SQL Anywhere 16 » UltraLite » Windows Sample Application.

3. On the File menu, click Synchronize.

   The application synchronizes and the MobiLink server messages window displays messages showing the synchronization taking place.

   The synchronization script determines which subset of customers, products, and orders is downloaded to the application when user 50 logs in. In this case, only orders that have not yet been approved are downloaded.

4. Confirm that the company name and a sample order appear in the application window.

Results

The CustDB application synchronizes with the consolidated database.
Lesson 3: Updating data in the UltraLite database

In this lesson, you use the CustDB application to add, update, and delete data in the remote database.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Building and running the CustDB application” on page 123.

Task

1. Browse orders.

   Browsing orders is accomplished by using a similar method for each version of the CustDB application. By browsing an order, you are scrolling through the data in your local UltraLite database. Because customers are sorted alphabetically, you can easily scroll through the list and locate a customer by name.
   a. To scroll down the list of customers, click Next.
   b. To scroll up through the list of customers, click Previous.

2. Add an order.

   Adding an order is carried out in a similar way in each version of the CustDB application. By adding an order, you modify the data in your local UltraLite database. This data is not shared with the consolidated database until you synchronize.
   a. Click Order » New.
   b. In the Customer list, click Basements R Us.
   c. In the Product list, click Screwmaster Drill. The price of this item is automatically entered in the Price field.
   d. In the Quantity field, type 20.
   e. In the Discount field, type 5 (percent) and click OK.

3. Approve, deny, and delete orders.

   Because you have authenticated your identity as user ID 50, you are a manager that can perform all the same tasks as a sales person, but you have the added ability to accept or reject orders. By accepting or rejecting an order, you change its status and add an additional note for the sales person to review. However, the data in the consolidated database is unchanged until you synchronize.
   a. Approve the order for Apple Street Builders.
      i. To locate the customer, click Previous.
ii. To approve the order, click Order and then Approve.
iii. In the Note list, click Good.
iv. Click OK.

The order appears with a status of Approved.

b. Deny the order for Art's Renovations.
i. Go to the next order in the list, which is from Art's Renovations.
ii. To deny the order, click Order and then Deny.
iii. In the Note list, click Discount Is Too High.
iv. Click OK.

The order appears with a status of Denied.

c. Delete the order for Awnings R Us.
i. Go to the next order in the list, which is from Awnings R Us.
ii. Delete this order by choosing Order » Delete.
iii. Click OK to confirm the deletion.

The order is marked as deleted. However, the current data remains in the UltraLite remote database until you synchronize changes to the consolidated database.

Results

Modifications to the data in the UltraLite database are saved but not synchronized with the CustDB database.

Next

Proceed to “Lesson 4: Synchronizing the UltraLite database with the consolidated database” on page 126.

See also

● “Tables in the CustDB databases” [MobiLink - Getting Started]

Lesson 4: Synchronizing the UltraLite database with the consolidated database

In this lesson, you synchronize databases and use either Interactive SQL or Sybase Central to connect to the consolidated database and confirm that your changes were synchronized.

Prerequisites

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Building and running the CustDB application” on page 123.
Context and remarks

The synchronization process for the CustDB application removes approved orders from your database.

Task

1. Synchronize the UltraLite database.

   From the File menu, click Synchronize Database.

2. Confirm that synchronization took place.

   At the remote database, you can confirm that all required transactions occurred by checking that the order for Awnings R Us is now deleted. Perform this action by browsing the orders to confirm the absence of this entry.

   At the consolidated database, you can also confirm that all required actions occurred by checking data.

   ● Confirm that synchronization took place by using Sybase Central.
     a. Click Start » Programs » SQL Anywhere 16 » Administration Tools » Sybase Central.
     b. Click Connections » Connect With SQL Anywhere 16.
     c. In the Action dropdown menu, click Connect With An ODBC Data Source.
     d. Click ODBC Data Source Name.
     e. Click Browse and click SQL Anywhere 16 CustDB.
     f. Click OK.
     g. Click Connect.
     h. Double-click Tables.
     i. Double-click ULOrder.
     j. Click the Data tab and verify that order 5100 is approved, order 5101 is denied, and order 5102 is deleted.

   ● Confirm that synchronization took place using Interactive SQL.
     a. Connect to the consolidated database from Interactive SQL.
        i. Click Start » Programs » SQL Anywhere 16 » Administration Tools » Interactive SQL.
        ii. In the Action dropdown list, click Connect With An ODBC Data Source.
        iii. Click ODBC Data Source Name and click SQL Anywhere 16 CustDB.
     b. To confirm that the approval and denial have been synchronized, execute the following statement:

        ```sql
        SELECT order_id, status
        FROM ULOrder
        WHERE status IS NOT NULL
        ```

        The results show that order 5100 is approved and 5101 is denied.
c. The deleted order has an order_id of 5102. The following query returns no rows, demonstrating that the order has been removed from the system:

```sql
SELECT *
FROM ULOrder
WHERE order_id = 5102
```

**Results**

The approved orders are removed from the database and you confirmed the removal.

**Next**

Proceed to “Lesson 5: Browsing MobiLink synchronization scripts” on page 128.

---

**Lesson 5: Browsing MobiLink synchronization scripts**

In this lesson, you browse synchronization scripts to get a better understanding of how the CustDB synchronization logic works.

**Prerequisites**

This lesson assumes you have completed all preceding lessons. See “Lesson 1: Building and running the CustDB application” on page 123.

**Task**

1. Click **Start** » **Programs** » **SQL Anywhere 16** » **Administration Tools** » **Sybase Central**.

2. In the left pane of **Sybase Central**, right-click **MobiLink 16** and then click **Open Project**.

3. Navigate to `%SQLANYSAMP16\MobiLink\CustDB` and click `project.mlp`.

4. Double-click **Consolidated Databases** and then double-click the CustDB consolidated database.

   You are connected to the consolidated database based on the connection information that was provided when you added the consolidated database to your project.

5. Double-click **Connection Scripts**.

   The right pane lists a set of synchronization scripts and a set of events with which these scripts are associated. As the MobiLink server carries out the synchronization process, it triggers a sequence of events. Any synchronization script associated with an event is run at that time. By writing synchronization scripts and assigning them to synchronization events, you can control the actions that are carried out during synchronization.

6. Click **Synchronized Tables**.
7. In the right pane, double-click **ULCustomer**.

A set of scripts specific to this table, and their corresponding events appears. These scripts control the way that data in the ULCustomer table is synchronized with the remote databases.

**Results**

You have reviewed the synchronization scripts.

**Next**

None.

**See also**

- “Synchronization script writing” [MobiLink - Server Administration]
- “Synchronization logic source code” [MobiLink - Getting Started]
- “Synchronization design” [MobiLink - Getting Started]
- “UltraLite clients” on page 63
- “Connection scripts” [MobiLink - Server Administration]
- “Table scripts” [MobiLink - Server Administration]
UltraLite database reference

This section provides a reference for UltraLite database properties, options, connection parameters, and utilities.

UltraLite creation parameters

[This topic has been updated for build 1823.]

This section describes the UltraLite creation parameters that are available when you create a new UltraLite database.

In addition to creation parameters, the following UltraLite connection parameters can be used as creation parameters:

- “UltraLite CE_FILE connection parameter” on page 161
- “UltraLite DB connection parameter”
- “UltraLite DBKEY connection parameter” on page 166
- “UltraLite desktop connection parameter” on page 168
- “UltraLite device connection parameter” on page 169
- “UltraLite MIRROR_FILE connection parameter” on page 170
- “UltraLite NT_FILE connection parameter” on page 171
- “UltraLite PWD connection parameter” on page 172
- “UltraLite RESERVE_SIZE connection parameter” on page 174
- “UltraLite UID connection parameter” on page 176

Members

UltraLite supports the following creation parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>case</td>
<td>Sets the case-sensitivity of string comparisons in the UltraLite database. See “UltraLite case creation parameter” on page 133.</td>
</tr>
<tr>
<td>checksum_level</td>
<td>Sets the level of checksum validation in the database. See “UltraLite checksum_level creation parameter” on page 134.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>collation</td>
<td>Sets the collation sequence used by the UltraLite database. Setting this property with or without the UTF-8 property determines the character set of the database. See “UltraLite character sets” on page 24 and “UltraLite collation creation parameter” on page 135 and “UltraLite utf8_encoding creation parameter” on page 155.</td>
</tr>
<tr>
<td>date_format</td>
<td>Sets the default string format in which dates are retrieved from the database. See “UltraLite date_format creation parameter” on page 136.</td>
</tr>
<tr>
<td>date_order</td>
<td>Controls the interpretation of date ordering of months, days, and years. See “UltraLite date_order creation parameter” on page 138.</td>
</tr>
<tr>
<td>fips</td>
<td>Controls the use of AES FIPS-certified encryption. See “Database security” on page 26 and “UltraLite fips creation parameter” on page 140.</td>
</tr>
<tr>
<td>max_hash_size</td>
<td>Sets the default index hash size in bytes. See “UltraLite max_hash_size creation parameter” on page 141.</td>
</tr>
<tr>
<td>nearest_century</td>
<td>Controls the interpretation of two-digit years in string-to-date conversions. See “UltraLite nearest_century creation parameter” on page 142.</td>
</tr>
<tr>
<td>obfuscate</td>
<td>Controls whether data in the database is obfuscated. Obfuscation is a form of simple encryption. See “Database security” on page 26 and “UltraLite obfuscate creation parameter” on page 143.</td>
</tr>
<tr>
<td>page_size</td>
<td>Defines the database page size. See “UltraLite page_size creation parameter” on page 144.</td>
</tr>
<tr>
<td>precision</td>
<td>Specifies the maximum number of digits in decimal point arithmetic results. See “UltraLite precision creation parameter” on page 146.</td>
</tr>
<tr>
<td>scale</td>
<td>Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision. See “UltraLite scale creation parameter” on page 147.</td>
</tr>
<tr>
<td>time_format</td>
<td>Sets the format for times retrieved from the database. See “UltraLite time_format creation parameter” on page 148.</td>
</tr>
<tr>
<td>timestamp_format</td>
<td>Sets the format for timestamps retrieved from the database. See “UltraLite timestamp_format creation parameter” on page 150.</td>
</tr>
<tr>
<td>timestamp_increment</td>
<td>Determines how the timestamp is truncated in UltraLite. See “UltraLite timestamp_increment creation parameter” on page 152.</td>
</tr>
</tbody>
</table>
### UltraLite timestamp_with_time_zone_format creation parameter

This option sets the format for TIMESTAMP WITH TIME ZONE values retrieved from the database. See “UltraLite timestamp_with_time_zone_format creation parameter” on page 153.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp_with_time_zone</td>
<td>This option sets the format for TIMESTAMP WITH TIME ZONE values retrieved from the database. See “UltraLite timestamp_with_time_zone_format creation parameter” on page 153.</td>
</tr>
</tbody>
</table>

### utf8_encoding creation parameter

Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode. See “UltraLite character sets” on page 24 and “UltraLite utf8_encoding creation parameter” on page 155.

### UltraLite case creation parameter

Sets the case sensitivity of string comparisons in the UltraLite database. Pass in `case=respect` to the creation string parameter of the CreateDatabase method in your programming interface (or `case=ignore` for a case-insensitive database.

**Syntax**

```
ulinit --case=value database.udb
```

**Allowed values**

Ignore, Respect

**Default**

Ignore

**Remarks**

The case sensitivity of data is reflected in tables, indexes, and so on. By default, UltraLite databases perform case-insensitive comparisons, although data is always held in the case in which you enter it. Identifiers (such as table and column names) and user IDs are always case insensitive, regardless of the database case sensitivity. Passwords are always case sensitive, regardless of the case sensitivity of the database.

The results of comparisons on strings, and the sort order of strings, depend in part on the case sensitivity of the database.

There are some collations where particular care is required when assuming case insensitivity of identifiers. In particular, Turkish collations have a case-conversion behavior that can cause unexpected and subtle errors. The most common error is that a system object containing a letter i or I is not found.

You cannot change the case of an existing database. Instead, you must create a new database.
From Sybase Central, you can set the case sensitivity in any wizard that creates a database. On the **New Database Collation And Character Set** page, click the **Use Case-sensitive String Comparisons** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

**See also**

- “Strings in UltraLite” on page 252
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite -.NET Programming]
- “Accessing creation parameter values” on page 23

**UltraLite checksum_level creation parameter**

Sets the level of checksum validation for the database.

**Syntax**

```
ulinit --checksum_level=value
```

**Allowed values**

- 0, 1, 2

**Default**

- 0

**Remarks**

Checksums are used to detect offline corruption on pages stored to disk, flash, or memory, which can help reduce the chances of other data being corrupted as the result of a bad critical page. Depending on the level you choose, UltraLite calculates and records a checksum for each database page before it writes the page to storage.

If the calculated checksum does not match the stored checksum for a page read from storage, the page has been modified or became corrupted during the storage/retrieval of the page. If a checksum validation fails, when the database loads a page, UltraLite stops the database and reports a fatal error. This error cannot be corrected; you must re-create your UltraLite database and report the database failure to iAnywhere.

If you unload and reload an UltraLite database with checksums enabled, the checksum level is preserved and restored.

The following values are supported for the checksum_level:

- **0** Do not add checksums to database pages.
Add checksums to important database pages, such as indexes and synchronization status pages, but not row pages.

Add checksums to all database pages.

From Sybase Central, you can configure the use of checksums in any wizard that creates a database. On the **New Database Storage Settings** page, click the **Checksum Level For Database Pages** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

**See also**

- checksum_level database property: “UltraLite database properties” on page 177
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “UltraLite performance tips” on page 439
- “UltraLite page_size creation parameter” on page 144
- “UltraLite database connections” on page 31

---

### UltraLite collation creation parameter

Sets the database collation.

**Syntax**

ulinit --collation=value

**Allowed values**

String

**Default**

1252Latin1

**Remarks**

You can also view a list of supported collations in UltraLite by executing the following command:

ulinit -Z

From Sybase Central, you can set the collation in any wizard that creates a database. On the **New Database Collation And Character Set** page, click either the default collation (1252Latin1), or an alternate one from the list.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.
See also

- “UltraLite supported collations” on page 25
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “UltraLite character sets” on page 24
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23

UltraLite date_format creation parameter

Sets the format for dates retrieved from the database.

For Android smartphones, you can use Connection.setOption(OPTION_DATE_FORMAT, value) as an alternative to setting this creation parameter. See “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming] and “Connection.OPTION_DATE_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming].

Syntax

ulinit --date_format=value

Allowed values

String

Default

YYYY-MM-DD (this corresponds to ISO date format specifications)

Remarks

DATE data type values are represented in a format set by the date_format creation parameter. Date values can, however, also be represented by strings. Before the value can be retrieved, it must be assigned to a string.

UltraLite builds a date from date parts. Date parts can include the year, the month, the day of the month, the day of the week, the day of the year, the hour, the minute, and the second (and parts thereof).

ISO (YYYY-MM-DD) is the default date format and order. For example, "7th of January 2006" in this international format is written: 2006-01-07. If you do not want to use the default ISO date format and order, you must specify a different format and order for these date parts.

The format is a string using the following symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YY</td>
<td>Two digit year.</td>
</tr>
</tbody>
</table>
### Symbol Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY</td>
<td>Four digit year.</td>
</tr>
<tr>
<td>MM</td>
<td>Two digit month, or two digit minutes if following a colon (as in hh:mm).</td>
</tr>
<tr>
<td>MMM[m...]</td>
<td>Character short form for months—as many characters as there are &quot;m&quot;s. An uppercase M causes the output to be made uppercase.</td>
</tr>
<tr>
<td>D</td>
<td>Single digit day of week, (0 = Sunday, 6 = Saturday).</td>
</tr>
<tr>
<td>DD</td>
<td>Two digit day of month. A leading zero is not required.</td>
</tr>
<tr>
<td>DDD[d...]</td>
<td>Character short form for day of the week. An uppercase D causes the output to be made uppercase.</td>
</tr>
<tr>
<td>HH</td>
<td>Two digit hours. A leading zero is not required.</td>
</tr>
<tr>
<td>NN</td>
<td>Two digit minutes. A leading zero is not required.</td>
</tr>
<tr>
<td>SS[s...]</td>
<td>Seconds and parts of a second.</td>
</tr>
<tr>
<td>AA</td>
<td>Use 12 hour clock. Indicate times before noon with AM.</td>
</tr>
<tr>
<td>PP</td>
<td>Use 12 hour clock. Indicate times after noon with PM.</td>
</tr>
<tr>
<td>JJJ</td>
<td>Day of the year, from 1 to 366.</td>
</tr>
</tbody>
</table>

You cannot change the date format of an existing database. Instead, you must create a new database.

Allowed values are constructed from the symbols listed in the table above. Each symbol is substituted with the appropriate data for the date that is being formatted.

For the character short forms, the number of letters specified is counted. The A.M. or P.M. indicator (which could be localized) is also truncated, if necessary, to the number of bytes corresponding to the number of characters specified.

**Controlling output case**  For symbols that represent character data (such as MMM), you can control the case of the output as follows:

- Type the symbol in uppercase to have the format appear in uppercase. For example, MMM produces JAN.
- Type the symbol in lowercase to have the format appear in lowercase. For example, mmm produces jan.
- Type the symbol in mixed case to have UltraLite choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.

**Controlling zero-padding**  For symbols that represent numeric data, you can control zero-padding with the case of the symbols:
○ Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.

○ Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.

From Sybase Central, you can set the date format in any wizard that creates a database. On the **New database creation parameters** page, click the **Date Format** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

**See also**
- “UltraLite date_order creation parameter” on page 138
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23

**Example**
The following table illustrates date_format settings, together with the output from a SELECT CURRENT DATE statement, executed on Thursday May 21, 2001.

<table>
<thead>
<tr>
<th>date_format syntax used</th>
<th>Result returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY/MM/DD/ddd</td>
<td>2001/05/21/thu</td>
</tr>
<tr>
<td>JJJ</td>
<td>141</td>
</tr>
<tr>
<td>mmm YYYY</td>
<td>may 2001</td>
</tr>
<tr>
<td>MM-YYYY</td>
<td>05-2001</td>
</tr>
</tbody>
</table>

**UltraLite date_order creation parameter**

Controls the interpretation of date formats.

For Android smartphones, you can use Connection.setOption(OPTION_DATE_ORDER, value) as an alternative to setting this creation parameter. See “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming] and “Connection.OPTION_DATE_ORDER variable [UltraLiteJ]” [UltraLite - Java Programming].

**Syntax**

ulinit --date_order=value
Allowed values

MDY, YMD, DMY

Default

YMD (this corresponds to ISO date format specifications)

Remarks

DATE data type values are represented in a format set by the date_format creation parameter. Date values can, however, also be represented by strings. Before the value can be retrieved, it must be assigned to a string.

UltraLite builds a date from date parts. Date parts can include the year, the month, the day of the month, the day of the week, the day of the year, the hour, the minute, and the second (and parts thereof).

ISO (YYYY-MM-DD) is the default date format and order. For example, "7th of January 2006" in this international format is written: 2006-01-07. If you do not want to use the default ISO date format and order, you must specify a different format and order for these date parts.

You cannot change the date order of an existing database. Instead, you must create a new database.

From Sybase Central, you can set the date order in any wizard that creates a database. On the New database creation parameters page, click the Date Order option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- “UltraLite date_format creation parameter” on page 136
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23

Example

Different values determine how the date of 10/11/12 is translated:

<table>
<thead>
<tr>
<th>Syntax used</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDY</td>
<td>Oct 11 1912</td>
</tr>
<tr>
<td>YMD</td>
<td>Nov 12 1910</td>
</tr>
<tr>
<td>DMY</td>
<td>Nov 10 1912</td>
</tr>
</tbody>
</table>
UltraLite fips creation parameter

Controls whether the new database should be encrypted using AES or AES_FIPS strong encryption. This parameter is not supported by UltraLiteJ, or UltraLite for iPhone.

Syntax

{ulinit -a | ulload -c} --fips=value --key=value

Allowed values

Yes (use AES_FIPS), No (use AES)

Default

Yes

Remarks

The only way to change the type of database encryption is to recreate the database with the appropriate fips or obfuscate creation parameter. You can change the database encryption key by specifying a new encryption key on the Connection object. Users connecting to the database must supply the key each time they connect.

From Sybase Central, you can configure encryption in any wizard that creates a database. On the New Database Storage Settings page, click the AES FIPS Algorithm option. You must also set and confirm the encryption key.

Set this parameter as one of the creation parameters for the create database method on the database manager class.

To deploy a database with the fips option turned on, copy all appropriate libraries for your platform.

See also

- “Simple encryption and strong encryption” [SQL Anywhere Server - Database Administration]
- “Database security” on page 26
- “UltraLite obfuscate creation parameter” on page 143
- “UltraLite DBKEY connection parameter” on page 166
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULChangeEncryptionKey method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “ULConnection.ChangeEncryptionKey method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23
UltraLite max_hash_size creation parameter

Sets the maximum default index hash size in bytes.

Syntax

```
ulinit --max_hash_size=value
```

Allowed values

0 to 32 bytes

Default

4 bytes

Remarks

A hash is an optional part of an index entry that is stored in the index page. The hash transforms the actual row values for the indexed columns into a numerical equivalent (a key), while still preserving ordering for that index. The size of the key, and how much of the actual value UltraLite hashes, is determined by the hash size you set.

A row ID allows UltraLite to locate the row for the actual data in the table. A row ID is always part of an index entry. If you set the hash size to 0 (disable index hashing), then the index entry only contains this row ID. For all other hash sizes, the hash key, which can contain all or part of the transformed data in that row, is stored along with the row ID in the index page. You can improve query performance on these indexed columns because UltraLite may not always need to find, load, and unpack data before it can compare actual row values.

Determining an appropriate default database hash size requires that you evaluate the trade-off between query efficiency and database size: the higher the maximum hash value, the larger the database size grows.

UltraLite only uses as many bytes as required for the data type(s) of the column(s), up to the maximum value specified by this parameter. The default hash size is only used if you do not set a size when you create the index. If you set the default hash size to 0, UltraLite does not hash row values.

You cannot change the hash size for an existing index. When creating a new index, you can override the default value with the UltraLite Create Index Wizard in Sybase Central, or with the WITH MAX SIZE clause of a CREATE INDEX or a CREATE TABLE statement.

If you declare your columns as DOUBLE, FLOAT, or REAL, no hashing is used. The hash size is always ignored.

From Sybase Central, you can set the maximum hash size in any wizard that creates a database. On the New Database Storage Settings page, click the Maximum Hash Size For Indexes option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.
UltraLite nearest_century creation parameter

Controls the interpretation of two-digit years in string-to-date conversions

For Android smartphones, you can use Connection.setOption(OPTION_NEAREST_CENTURY, value) as an alternative to setting this creation parameter. See “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming] and “Connection.OPTION_NEAREST_CENTURY variable [UltraLiteJ]” [UltraLite - Java Programming].

Syntax

ulinit --nearest_century=value

Allowed values

Integer, between 0 and 100, inclusive

Default

50

Remarks

UltraLite automatically converts a string into a date when a date value is expected, even if the year is represented in the string by only two digits. For a two-digit date, you need to set the appropriate rollover value. Two digit years less than the value are converted to 20yy, while years greater than or equal to the value are converted to 19yy.

Choosing an appropriate rollover value typically is determined by:

- The use of two-digit dates  Otherwise, nearest century conversion isn't applicable. Two-digit years less than the nearest_century value you set are converted to 20yy, while years greater than or equal to the value are converted to 19yy.

  It is recommended that you store four-digit dates to avoid issues with incorrect conversions.
● **Consolidated database compatibility**  For example, the historical SQL Anywhere behavior is to add 1900 to the year. Adaptive Server Enterprise behavior is to use the nearest century, so for any year where value $yy$ is less than 50, the year is set to 20$yy$.

● **What the date represents: past event or future event**  Birth years are typically those that would require a lower rollover value since they occur in the past. So for any year where $yy$ is less than 20, the year should be set to 20$yy$. However, if the date is used as an expiry date, then having a higher value would be a logical choice, since the date is occurring in the future.

You cannot change the nearest century of an existing database. Instead, you must create a new database.

From Sybase Central, you can configure the nearest century setting in any wizard that creates a database. On the **New database creation parameters** page, click the **Nearest Century** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

**See also**

- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23

### UltraLite obfuscate creation parameter

Controls obfuscation of data in the database.

**Syntax**

```
{ulinit -a | ulload -c } obfuscate=value
```

**Allowed values**

Boolean.

**Default**

0 (databases are not obfuscated)

**Remarks**

Obfuscation is a form of simple encryption that makes it more difficult for someone using a disk utility to look at the file to decipher the data in your database. Simple encryption does not require a key to encrypt the database.

You must use strong encryption to make the database inaccessible without the correct encryption key.
From Sybase Central, you can set obfuscation in any wizard that creates a database. On the **New Database Storage Settings** page, click the **Use Simple Encryption (Obfuscation)** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

**See also**

- “Database security” on page 26
- “UltraLite fips creation parameter” on page 140
- “UltraLite DBKEY connection parameter” on page 166
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [*UltraLite - C and C++ Programming*]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [*UltraLite - C and C++ Programming*]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [*UltraLite - .NET Programming*]
- “Accessing creation parameter values” on page 23

---

**UltraLite page_size creation parameter**

Defines the database page size in kilobytes.

**Syntax**

```
{ulinit | ulload} -c --page_size=sizeK
```

**Allowed values**

1K, 2K, 4K, 8K, 16K

**Default**

4K

**Remarks**

The page size must be entered with a K or k after the digit, or, alternatively, the equivalent number of bytes (1024, 2048, 4096, 8192, or 16384).

UltraLite databases are stored in pages, and all I/O operations are carried out a page at a time. The page size you choose can affect the performance or size of the database.

If you use any value other than those listed, the size is changed to the next larger size. If you do not specify a unit, bytes are assumed.

If your platform has limited dynamic memory, consider using a smaller page size to limit the effect on synchronization memory requirements.

When choosing a page size, you should keep the following guidelines in mind:
- **Database size**  Larger databases usually benefit from a larger page size. Larger pages hold more information and therefore use space more effectively—particularly if you insert rows that are slightly more than half a page in size. The larger the page, the less page swapping that is required.

- **Number of rows**  Because a row (excluding BLOBs) must fit on a page, the page size determines how large the largest packed row can be, and how many rows you can store on each page. Sometimes reading one page to obtain the values of one row may have the side effect of loading the contents of the next few rows into memory.

- **Query types**  In general, smaller page sizes are likely to benefit queries that retrieve a relatively small number of rows from random locations. By contrast, larger pages tend to benefit queries that perform sequential table scans.

- **Cache size**  Large page sizes may require larger cache sizes. With dynamic cache sizing, UltraLite grows the cache as required.

- **Index entries**  Page size also affects indexes. The larger the database page, the more index entries it can hold.

- **Device memory**  Small pages are particularly useful if your database must run on small devices with limited memory. For example, 1 MB of memory can hold 1000 pages that are each 1 KB in size, but only 250 pages that are 4 KB in size.

You cannot change the page size of an existing database. Instead, you must create a new database.

From Sybase Central, you can set the page size in any wizard that creates a database. On the **New Database Storage Settings** page, click the appropriate byte value.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

**See also**
- “Row packing and table definitions” on page 46
- “UltraLite indexes” on page 52
- “UltraLite case creation parameter” on page 133
- “UltraLite CACHE_SIZE connection parameter” on page 160
- “UltraLite CACHE_MIN_SIZE connection parameter” on page 159
- “UltraLite CACHE_MAX_SIZE connection parameter” on page 158
- “UltraLite RESERVE_SIZE connection parameter” on page 174
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite -.NET Programming]
- “Accessing creation parameter values” on page 23

**Example**

To set the page size of the database to 8 KB, specify `page_size=8k` or `page_size=8192`: 

---

---
ulinit test.udb -a --page_size=8k

UltraLite precision creation parameter

Specifies the maximum number of digits in decimal point arithmetic results.

For Android smartphones, you can use Connection.setOption(OPTION_PRECISION, value) as an alternative to setting this creation parameter. See “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming] and “Connection.OPTION_PRECISION variable [UltraLiteJ]” [UltraLite - Java Programming].

Syntax

```
ulinit precision=value
```

Allowed values

Integer, between 1 and 127, inclusive

Default

30

Remarks

The position of the decimal point is determined by the precision and the scale of the number: precision is the total number of digits to the left and right of the decimal point; scale is the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

Choosing an appropriate decimal point position is typically determined by:

- **The type of arithmetic procedures you perform**  
  Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision.

  For example, when a DECIMAL(8,2) is multiplied with a DECIMAL(9,2), the result could require a DECIMAL(17,4). If precision is 15, only 15 digits are kept in the result. If scale is 4, the result is a DECIMAL(15,4). If scale is 2, the result is a DECIMAL(15,2). In both cases, there is a possibility of an overflow error.

- **The relationship between scale and precision values**  
  The scale sets the number of digits in the fractional part of the number, and cannot be negative or greater than the precision.

You cannot change the precision of an existing database. Instead, you must create a new database.

If you are using an Oracle database as the consolidated database, all UltraLite remotes and the Oracle consolidated database must have the same precision value.

From Sybase Central, you can set the precision in any wizard that creates a database. On the New database creation parameters page, click the Precision option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.
See also

- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23

UltraLite scale creation parameter

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

For Android smartphones, you can use Connection.setOption(OPTION_SCALE, value) as an alternative to setting this creation parameter. See “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming] and “Connection.OPTION_SCALE variable [UltraLiteJ]” [UltraLite - Java Programming].

Syntax

ulinit --scale=value

Allowed values

Integer, between 0 and 127, inclusive

Default

6

Remarks

The position of the decimal point is determined by the precision and the scale of the number: precision is the total number of digits to the left and right of the decimal point; scale is the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

Choosing an appropriate decimal point position is typically determined by:

- **The type of arithmetic procedures you perform** Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum precision.

  For example, when a DECIMAL(8,2) is multiplied with a DECIMAL(9,2), the result could require a DECIMAL(17,4). If precision is 15, only 15 digits are kept in the result. If scale is 4, the result is a DECIMAL(15,4). If scale is 2, the result is a DECIMAL(15,2). In both cases, there is a possibility of an overflow error.

- **The relationship between scale and precision values** The scale sets the number of digits in the fractional part of the number, and cannot be negative or greater than the precision.

  You cannot change the scale of an existing database. Instead, you must create a new database.
From Sybase Central, you can set the scale in any wizard that creates a database. On the New database creation parameters page, click the Scale option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also
- “UltraLite precision creation parameter” on page 146
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23

Example
When a DECIMAL(8,2) is multiplied with a DECIMAL(9,2), the result could require a DECIMAL(17,4). If precision is 15, only 15 digits are kept in the result. If scale is 4, the result is DECIMAL(15,4). If scale is 2, the result is a DECIMAL(15,2). In both cases, there is a possibility of overflow.

UltraLite time_format creation parameter
Sets the format for times retrieved from the database.

For Android smartphones, you can use Connection.setOption(OPTION_TIME_FORMAT, value) as an alternative to setting this creation parameter. See “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming] and “Connection.OPTION_TIME_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming].

Syntax
ulinit --time_format=value

Allowed values
String (composed of the symbols listed below)

Default
HH:NN:SS.SSS

Remarks
UltraLite writes times from time parts you set with the time_format creation parameter. Time parts can include hours, minutes, seconds, and milliseconds.

Time values can also be represented by strings. Before a time value can be retrieved, it must be assigned to a string variable.
ISO (HH:MM:SS) is the default time format. For example, "midnight" in this international format is written: 00:00:00. If you do not want to use the default ISO time format, you must specify a different format and order for these time parts.

The format is a string using the following symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HH</td>
<td>Two digit hours (24 hour clock).</td>
</tr>
<tr>
<td>NN</td>
<td>Two digit minutes.</td>
</tr>
<tr>
<td>MM</td>
<td>Two digit minutes if following a colon (as in HH:MM).</td>
</tr>
<tr>
<td>SS.SSS</td>
<td>Two digit seconds plus fraction.</td>
</tr>
</tbody>
</table>

You cannot change the time format of an existing database. Instead, you must create a new database.

Each symbol is substituted with the appropriate data for the time that is being formatted. Any format symbol that represents character rather than digit output can be put in uppercase, which causes the substituted characters to be in uppercase. For numbers, using mixed case in the format string suppresses leading zeros.

You can control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as HH or hh) to allow zero padding. For example, HH:NN:SS could produce 01:01:01.

- Type the symbol in mixed case (such as Hh or hH) to suppress zero padding. For example, Hh:Nn:Ss could produce 1:1:1.

From Sybase Central, you can set the time format in any wizard that creates a database. On the **New database creation parameters** page, click the **Time Format** option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

**See also**

- “UltraLite timestamp_format creation parameter” on page 150
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23
Example

If a transaction was executed at 3:30 P.M. and you used the default time_format syntax of HH:NN:SS.SSS, the result would be:

15:30:55.0

UltraLite timestamp_format creation parameter

Sets the format for timestamps that are retrieved from the database.

For Android smartphones, you can use Connection.setOption(OPTION_TIMESTAMP_FORMAT, value) as an alternative to setting this creation parameter. See “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming] and “Connection.OPTION_TIME_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming].

Syntax

ulinit --timestamp_format=value

Allowed values

String

Default

YYYY-MM-DD HH:NN:SS.SSS

Remarks

UltraLite creates a timestamp from date and time parts that you set with the date_format and time_format creation parameters. Together, date and time total seven parts (year, month, day, hour, minute, second, and millisecond).

Timestamp values can also be represented by strings. Before it can be retrieved, a timestamp value must be assigned to a string variable.

Typically timestamp columns ensure that data integrity is maintained when synchronizing with a consolidated database. Timestamps help identify when concurrent data updates have occurred among multiple remote databases by tracking the last time that each user synchronized.

Tip

Ensure that the consolidated database and the UltraLite remote database maintain timestamps and timestamp increments to the same resolution. By setting these creation parameters to match that of the consolidated database, you can help avoid spurious inequalities.

The format is a string using the following symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YY</td>
<td>Two digit year.</td>
</tr>
</tbody>
</table>
### Symbol Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY</td>
<td>Four digit year.</td>
</tr>
<tr>
<td>MM</td>
<td>Two digit month, or two digit minutes if following a colon (as in HH:MM).</td>
</tr>
<tr>
<td>MMM[m...]</td>
<td>Character short form for months—as many characters as there are &quot;m&quot;s. An uppercase M causes the output to be made uppercase.</td>
</tr>
<tr>
<td>D</td>
<td>Single digit day of week, (0 = Sunday, 6 = Saturday).</td>
</tr>
<tr>
<td>DD</td>
<td>Two digit day of month. A leading zero is not required.</td>
</tr>
<tr>
<td>DDD[d...]</td>
<td>Character short form for day of the week. An uppercase D causes the output to be made uppercase.</td>
</tr>
<tr>
<td>HH</td>
<td>Two digit hours. A leading zero is not required.</td>
</tr>
<tr>
<td>NN</td>
<td>Two digit minutes. A leading zero is not required.</td>
</tr>
<tr>
<td>SS.SSS</td>
<td>Seconds and parts of a second.</td>
</tr>
<tr>
<td>AA</td>
<td>Use 12 hour clock. Indicate times before noon with AM.</td>
</tr>
<tr>
<td>PP</td>
<td>Use 12 hour clock. Indicate times after noon with PM.</td>
</tr>
<tr>
<td>JJJ</td>
<td>Day of the year, from 1 to 366.</td>
</tr>
</tbody>
</table>

You cannot change the timestamp format of an existing database. Instead, you must create a new database.

Allowed values are constructed from the symbols listed in the table above. Each symbol is substituted with the appropriate data for the date that is being formatted.

For the character short forms, the number of letters specified is counted. The A.M. or P.M. indicator (which could be localized) is also truncated, if necessary, to the number of bytes corresponding to the number of characters specified.

For symbols that represent character data (such as MMM), you can control the case of the output as follows:

- Type the symbol in all uppercase to have the format appear in all uppercase. For example, MMM produces JAN.
- Type the symbol in all lowercase to have the format appear in all lowercase. For example, mmm produces jan.
- Type the symbol in mixed case to have UltraLite choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.
For symbols that represent numeric data, you can control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.

- Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.

From Sybase Central, you can set the timestamp format in any wizard that creates a database. On the New database creation parameters page, click the Timestamp Format option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also
- “UltraLite timestamp_increment creation parameter” on page 152
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23
- “Implementing timestamp-based downloads” [MobiLink - Server Administration]
- “UltraLite concurrency” on page 452

Example
If a transaction was executed on Friday May 12, 2006 at 3:30 PM and you used the default timestamp_format syntax of YYYY-MM-DD HH:NN:SS.SSS, the result would be:

2006-05-12 15:30:55.0

**UltraLite timestamp_increment creation parameter**

Limits the resolution of timestamp values. As timestamps are inserted into the database, UltraLite truncates them to match this increment.

For Android smartphones, you can use Connection.setOption(OPTION_TIMESTAMP_INCREMENT, value) as an alternative to setting this creation parameter. See “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming] and “Connection.OPTION_TIMESTAMP_INCREMENT variable [UltraLiteJ]” [UltraLite - Java Programming].

Syntax

```bash
ulinit timestamp_increment=value
```

Allowed values

1 to 60000000 microseconds
Default

1 microsecond

Remarks

1000000 microseconds equals 1 second.

You cannot change the timestamp increment of an existing database. Instead, you must create a new database.

This increment is useful when a DEFAULT TIMESTAMP column is being used as a primary key or row identifier.

From Sybase Central, you can set the timestamp increment in any wizard that creates a database. On the New database creation parameters page, click the Timestamp Increment option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also

- “UltraLite timestamp_format creation parameter” on page 150
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULCreateDatabase method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite .NET Programming]
- “Accessing creation parameter values” on page 23
- “Implementing timestamp-based downloads” [MobiLink - Server Administration]
- “UltraLite concurrency” on page 452

Example

To store a value such as '2000/12/05 10:50:53:700', set this creation parameter to 100000. This value truncates the timestamp after the first decimal place in the seconds component.

UltraLite timestamp_with_time_zone_format creation parameter

Sets the format for TIMESTAMP WITH TIME ZONE values retrieved from the database.

Syntax

ulinit --timestamp_with_time_zone_format=value

Allowed values

String (composed of the symbols listed below)
Default

YYYY-MM-DD HH:NN:SS.SSS+HH:NN

Remarks

The format is a string using the following symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YY</td>
<td>Two digit year</td>
</tr>
<tr>
<td>YYYY</td>
<td>Four digit year</td>
</tr>
<tr>
<td>MM</td>
<td>Two digit month, or two digit minutes if following a colon (as in HH:MM)</td>
</tr>
<tr>
<td>MMM[m...]</td>
<td>Character short form for months—as many characters as there are &quot;m&quot;s</td>
</tr>
<tr>
<td>DD</td>
<td>Two digit day of month</td>
</tr>
<tr>
<td>DDD[d...]</td>
<td>Character short form for day of the week</td>
</tr>
<tr>
<td>HH</td>
<td>Two digit hours</td>
</tr>
<tr>
<td>NN</td>
<td>Two digit minutes</td>
</tr>
<tr>
<td>SS.SSSSSS</td>
<td>Seconds and fractions of a second, up to six decimal places. Not all platforms support timestamps to a precision of six places.</td>
</tr>
<tr>
<td>AA</td>
<td>A.M. or P.M. (12 hour clock)—omit AA and PP for 24 hour time</td>
</tr>
<tr>
<td>PP</td>
<td>P.M. if needed (12 hour clock)—omit AA and PP for 24 hour time</td>
</tr>
<tr>
<td>HH</td>
<td>Two digit hours (time zone offset)</td>
</tr>
<tr>
<td>NN</td>
<td>Two digit minutes (time zone offset)</td>
</tr>
</tbody>
</table>

Each symbol is substituted with the appropriate data for the date that is being formatted.

For symbols that represent character data (such as MMM), you can control the case of the output as follows:

- Type the symbol in all uppercase to have the format appear in all uppercase. For example, MMM produces JAN.
- Type the symbol in all lowercase to have the format appear in all lowercase. For example, mmm produces jan.
- Type the symbol in mixed case to have UltraLite choose the appropriate case for the language that is being used. For example, in English, typing Mmm produces May, while in French it produces mai.
If the character data is multibyte, the length of each symbol reflects the number of characters, not the number of bytes. For example, the MMM symbol specifies a length of three characters for the month.

For symbols that represent numeric data, you can control zero-padding with the case of the symbols:

- Type the symbol in same-case (such as MM or mm) to allow zero padding. For example, yyyy/mm/dd could produce 2002/01/01.

- Type the symbol in mixed case (such as Mm) to suppress zero padding. For example, yyyy/Mm/Dd could produce 2002/1/1.

**Note**

If you change the setting for timestamp_with_time_zone_format option in a way that re-orders the date format, be sure to change the date_order option to reflect the same change, and vice versa. See “date_order option” [SQL Anywhere Server - Database Administration].

**See also**

- “TIMESTAMP WITH TIME ZONE data type” [SQL Anywhere Server - SQL Reference]

---

### UltraLite utf8_encoding creation parameter

Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode.

**Syntax**

`ulload - c --utf8_encoding=value`

**Values**

Boolean.

**Default**

1 (databases are UTF-8 encoded)

**Remarks**

UTF-8 characters are represented by one to four bytes. For other multibyte collations, one or two bytes are used. For all provided multibyte collations, characters of two or more bytes are considered to be alphabetic. You can use these characters in identifiers without requiring double quotes.

Characters in an UltraLite database are either from the codepage implicit in the chosen collation, or are UTF8 encoded. UltraLite databases that use the UTF8BIN collation are automatically UTF8 encoded. If the operating system to which you are deploying your UltraLite application uses UTF8 or Unicode (like most Linux distributions, Windows Mobile and iPhone) or if you plan to store characters from multiple languages in your database, you should create your database using a UTF8 encoding. If you try synchronizing UTF-8 encoded characters into a consolidated table that does not support Unicode, a user error is reported.
From Sybase Central, you can choose UTF-8 encoding in any wizard that creates a database. On the New database collation and character set page, click the Yes, use UTF-8 as the database character set option.

From a client application, set this parameter as one of the creation parameters for the create database method on the database manager class.

See also
- “UltraLite platform requirements for character set encoding” on page 24
- “UltraLite character sets” on page 24
- “UltraLite Initialize Database utility (ulinit)” on page 198
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “ULDatabaseManager.CreateDatabase method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.CreateDatabase method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Accessing creation parameter values” on page 23

UltraLite connection parameters

This section describes the UltraLite connection parameters that are available when you connect to an UltraLite database.

Members

UltraLite supports the following connection parameters:

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACHE_MAX_SIZE,</td>
<td>Defines the size of the database cache. See “UltraLite CACHE_SIZE connection parameter” on page 160, “UltraLite CACHE_MIN_SIZE connection parameter” on page 159, and “UltraLite CACHE_MAX_SIZE connection parameter” on page 158.</td>
</tr>
<tr>
<td>CACHE_MIN_SIZE,</td>
<td></td>
</tr>
<tr>
<td>CACHE_SIZE</td>
<td></td>
</tr>
<tr>
<td>COMMIT_FLUSH</td>
<td>Determines when committed transactions are flushed to storage after a commit call. See “UltraLite COMMIT_FLUSH connection parameter” on page 163.</td>
</tr>
<tr>
<td>CON</td>
<td>Specifies a name of the current connection. See “UltraLite CON connection parameter” on page 164.</td>
</tr>
<tr>
<td>Parameter name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| CE_FILE, DBF, desktop, device, and NT_FILE | At creation time, these parameters set the location of the database. For subsequent connections, they tell UltraLite where to find the file. You can use DBF if you are creating a single-platform application or are connecting to an UltraLite administration tool. Use the other platform-specific versions if you are programming an UltraLite client that connects to different platform-specific databases. See:  
  - “UltraLite DBF connection parameter” on page 165  
  - “UltraLite CE_FILE connection parameter” on page 161  
  - “UltraLite desktop connection parameter” on page 168  
  - “UltraLite device connection parameter” on page 169  
  - “UltraLite NT_FILE connection parameter” on page 171 |
<p>| DBKEY                              | At creation-time, this parameter sets the encryption key used to encrypt the database. For subsequent connections, it specifies the encryption key used to encrypt the database. See “UltraLite DBKEY connection parameter” on page 166. |
| DBN                                | Identifies a running database by name rather than file name. See “UltraLite DBN connection parameter” on page 167.                                                                                       |
| MIRROR_FILE                        | Specifies the name of a database mirror file. See “UltraLite MIRROR_FILE connection parameter” on page 170.                                                                                                  |
| PWD                                | At creation-time, this parameter sets the initial password for a user. For subsequent connections, it supplies the password for the user ID. See “UltraLite PWD connection parameter” on page 172. |
| RESERVE_SIZE                       | Pre-allocates the file system space required for your UltraLite database without actually inserting any data. See “UltraLite RESERVE_SIZE connection parameter” on page 174.                        |
| START                              | Specifies the location of the UltraLite engine executable. See “UltraLite START connection parameter” on page 175.                                                                                       |
| TEMP_DIR                           | Specifies the name of the directory (which must already exist) into which UltraLite will place a temporary file (with a name derived from the database name). See “UltraLite TEMP_DIR connection parameter” on page 175. |</p>
<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID</td>
<td>At creation time, this parameter sets the initial user ID. For subsequent connections, it identifies a user to the database. The user ID must be one of up to four user IDs stored in the Ultra-Lite database. See “UltraLite UID connection parameter” on page 176.</td>
</tr>
</tbody>
</table>

See also
- “UltraLite connection strings and parameters” on page 31

**UltraLite CACHE_MAX_SIZE connection parameter**

Defines the maximum size of the database cache. UltraLite manages the cache size automatically, so setting this parameter should not be necessary.

Syntax

```
CACHE_MAX_SIZE=number{ k | m }
```

Default

The default maximum cache size is 20 MB for devices and 50 MB for desktops.

Remarks

The cache_max_size connection parameter specifies the maximum amount of memory to allocate for the file cache. By default, the size is in bytes. Use k or m to specify units of kilobytes or megabytes.

If you exceed the maximum cache size, your platform's upper cache size limit is used instead. UltraLite does not grow the cache size beyond the actual file size of the database.

If you specify a cache size limit that is greater than the size of your database, the excess space may be used for caching rows.

Any leading or trailing spaces in connection parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.
See also

- “UltraLite CACHE_MIN_SIZE connection parameter” on page 159
- “UltraLite CACHE_SIZE connection parameter” on page 160
- “UltraLite cache_allocation option” on page 181
- “UltraLite connection strings and parameters” on page 31
- “UltraLite page_size creation parameter” on page 144
- “UltraLite RESERVE_SIZE connection parameter” on page 174
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]
- “Cache size adjustment for an UltraLite database” on page 439

Example

The following connection string fragment sets the maximum cache size to 100 MB.

"CACHE_MAX_SIZE=100m"

UltraLite CACHE_MIN_SIZE connection parameter

Defines the minimum size of the database cache. UltraLite manages the cache size automatically, so setting this parameter should not be necessary.

Syntax

CACHE_MIN_SIZE=number{ k | m }

Default

The default cache size for devices is 256 KB. The default cache size for desktops is 512 KB.

Remarks

The cache_min_size connection parameter specifies the minimum amount of memory to allocate for the file cache. By default, the size is in bytes. Use k or m to specify units of kilobytes or megabytes.

If you set the minimum cache size to be greater than the maximum cache size, UltraLite returns an error message and the connection fails.

Any leading or trailing spaces in connection parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.
See also

- “UltraLite CACHE_MAX_SIZE connection parameter” on page 158
- “UltraLite CACHE_SIZE connection parameter” on page 160
- “UltraLite cache_allocation option” on page 181
- “UltraLite connection strings and parameters” on page 31
- “UltraLite page_size creation parameter” on page 144
- “UltraLite RESERVE_SIZE connection parameter” on page 174
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Example

The following connection string fragment sets the minimum cache size to 1 MB.

"CACHE_MIN_SIZE=1m"

---

**UltraLite CACHE_SIZE connection parameter**

Defines the initial size of the database cache. UltraLite manages the cache size automatically, so setting this parameter should not be necessary.

For Android smartphones, you can use Configuration.setPageSize as an alternative to setting this connection parameter. See “Configuration.setPageSize method [UltraLiteJ]” [UltraLite - Java Programming].

**Syntax**

```
CACHE_SIZE = number{ k | m }
```

**Default**

The default initial cache size is determined by the amount of memory available on your system and the size of the database.

**Remarks**

The cache_size connection parameter specifies the initial amount of memory to allocate for the file cache. This cache is used to hold recently-used pages from the database file in memory so they can be accessed quickly when needed again, and also to collect multiple modifications to a page before writing it back to storage. Accessing a page from the cache is many times faster than reading from storage. Writing to storage is more expensive, so grouping multiple modifications in a single write is important for performance. Encrypted databases also benefit from the cache because decryption occurs only when the page is loaded into the cache, and encryption occurs before the page is written back to storage. If the cache is sufficiently large, the overhead of encryption becomes negligible.
As an example of cache usage, consider synchronization. While UltraLite is receiving a download, the 
rows are inserted into the database, and referential integrity checks are performed. When inserted, the 
rows are also indexed; they are added to each index on the table. So, while synchronizing, the cache tends 
to hold the pages where the new rows are stored, as well as the index pages for the current table. 
Synchronization performance depends greatly on whether the cache is large enough to contain an 
appropriate working set of pages for a table being synchronized. If the cache is too small, row inserts may 
require repeated reads of index pages from storage, incurring a noticeable performance penalty over the 
case when the required index pages fit in the cache.

By default, the size is in bytes. Use k or m to specify units of kilobytes or megabytes.

If you exceed the permissible maximum cache size, it is automatically replaced with your platform's 
upper cache size limit.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot 
include leading single quotes, leading double quotes, or semicolons.

See also
- “UltraLite CACHE_MAX_SIZE connection parameter” on page 158
- “UltraLite CACHE_MIN_SIZE connection parameter” on page 159
- “UltraLite cache_allocation option” on page 181
- “UltraLite connection strings and parameters” on page 31
- “UltraLite page_size creation parameter” on page 144
- “UltraLite RESERVED_SIZE connection parameter” on page 174
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Example

The following connection string fragment sets the cache size to 20 MB.

"CACHE_SIZE=20m"

UltraLite CE_FILE connection parameter

When creating a new UltraLite database, this connection parameter names the new database file.

When opening a connection to an existing database, it identifies the database.

For Android smartphones, you can use Configuration.setDatabaseName as an equivalent to setting this 
connection parameter. See “Configuration.setDatabaseName method [UltraLiteJ]” [UltraLite - Java Programming].
Syntax

\[ \text{CE\_FILE} = \text{path}\backslash\text{ce\_db} \]

Default

DBF connection parameter.

Behavior

1. If DBF is specified, look for a matching database (identical filename) and connect if found, proceed to auto-start if not.

2. A database is auto-started when required if DBF is specified.

Remarks

You should use the CE\_FILE connection parameter for UltraLite client applications that use the same connection string to connect to a Microsoft Windows Mobile device, and other platforms.

The CE\_FILE connection parameter takes precedence over the DBF connection parameter. If you are connecting from an UltraLite administration tool, or your connection object only connects to a Windows Mobile database, use the DBF connection parameter.

The value of CE\_FILE must meet the file name requirements for Windows Mobile. If you include an absolute path to the database, then all directories must exist before setting the path to this file. UltraLite does not create them automatically.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

See also

- “UltraLite connection strings and parameters” on page 31
- “UltraLite file path formats in connection parameters” on page 33
- “Precedence of connection parameters for UltraLite administration tools” on page 31
- “UltraLite DBF connection parameter” on page 165
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Example

The following example creates a new connection and identifies different database files for the Windows desktop and Windows Mobile platforms:

```
Set Connection = DatabaseMgr.OpenConnection("DBF=d:\Dbfile.udb;CE\_FILE=myapp\MyDB.udb")
```
UltraLite COMMIT_FLUSH connection parameter

Determines when committed transactions are flushed to storage after a commit call. If no calls to commit are made by the UltraLite application, no flush can occur.

Syntax

\texttt{COMMIT\_FLUSH=\{ immediate | grouped | on\_checkpoint \}}

Default

immediate

Remarks

This connection parameter defines which transactions are recovered following a hardware failure or crash. You can group logical autocommit operations as a single recovery point.

By grouping these operations, you can improve UltraLite performance, but at the expense of data recoverability. There is a slight chance that a transaction may be lost—even though it has been committed—if a hardware failure or crash occurs after a commit, but before the transaction is flushed to storage.

The following parameters are supported:

- **immediate**  Committed transactions are flushed to storage immediately upon a commit call before the commit operation completes.

- **grouped**  Committed transactions are flushed to storage on a commit call, but only after a threshold you configure has been reached. You can configure either a transaction count threshold with the commit\_flush\_count database option or a time-based threshold with the commit\_flush\_timeout database option.

  If set, both the commit\_flush\_count and the commit\_flush\_timeout options act as possible triggers for the commit flush; the first threshold that is met triggers the flush. When the flush occurs, UltraLite sets the counter and the timer back to 0. Then, both the counter and timer are monitored, until one of these thresholds is reached again.

- **on\_checkpoint**  Committed transactions are flushed to storage on a checkpoint operation. You can perform a checkpoint with any of the following:

  - The CHECKPOINT statement. APIs that do not have a checkpoint method must use this SQL statement.
  - The ULCheckpoint function for UltraLite embedded SQL.
  - The Checkpoint method on a connection object in a C++ component.
UltraLite CON connection parameter

Names a connection so that switching to it is easier in multi-connection applications.

Syntax

CON = name

Default

No connection name.

Remarks

The CON connection parameter is global to the application.

Do not use this connection parameter unless you are going to establish and switch between two or more concurrent connections.

The connection name is not the same as the database name.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

See also

- “UltraLite connection strings and parameters” on page 31
- “UltraLite DBN connection parameter” on page 167
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite -.NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite -.NET Programming]

Example

The following connection string fragment sets the first connection name to MyFirstCon.

"CON=MyFirstCon"
UltraLite DBF connection parameter

When creating a new UltraLite database, this connection parameter names the new database file.

When opening a connection to an existing database, it indicates which database file you want to load and connect to.

Syntax

\texttt{DBF=ul-db}

Behavior

1. On connect, look to see if the database is already running. If DBN is specified, look for a matching database and connect if found, proceed to auto-start if not.

2. If DBF is specified, look for a matching database (identical filename) and connect if found, proceed to auto-start if not.

3. If neither DBN nor DBF is specified, and a single database is running, connect to it.

4. A database is auto-started when required if DBF is specified. If DBN is also specified, it becomes the name of the running database, otherwise a name is generated from the base filename.

Remarks

Because they are aliases, if DBF is used concurrently, the last one specified takes precedence.

If you are connecting to multiple databases on different devices from a single connection string, you can use the following parameters to name platform-specific alternates:

- CE\_FILE
- desktop
- device
- NT\_FILE

If specified, these platform-specific connection parameters take precedence over DBF.

The value of DBF must meet the file name requirements for the platform.

\textbf{Windows Mobile} If you are deploying to a Windows Mobile device, UltraLite utilities and wizards can administer an UltraLite database on an attached Windows Mobile device. To identify a file on a Windows Mobile device, you must specify the required absolute path, and use the \texttt{wce:} prefix.

Any leading or trailing spaces in parameter values are ignored. The value cannot include leading single quotes, leading double quotes, or semicolons.
See also

- “UltraLite connection strings and parameters” on page 31
- “UltraLite file path formats in connection parameters” on page 33
- “Precedence of connection parameters for UltraLite administration tools” on page 31
- “UltraLite DBN connection parameter” on page 167
- “UltraLite CE_FILE connection parameter” on page 161
- “UltraLite NT_FILE connection parameter” on page 171
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Examples

To connect to the database, *MyULdb.udb*, installed in the desktop directory *c:\mydb*, use the following connection string:

```
"DBF=c:\mydb\MyULdb.udb"
```

To connect to the same database that is deployed to the *UltraLite* folder of the attached Windows Mobile device, use the following connection string:

```
"DBF=wce:\UltraLite\MyULdb.udb"
```

### UltraLite DBKEY connection parameter

When creating a new UltraLite database, this connection parameter provides an encryption key for the database.

When opening a connection to an existing database, it provides the encryption key for the database.

**Syntax**

```
DBKEY=string
```

**Default**

No key is provided.

**Remarks**

If you do not specify the correct encryption key for the database, the connection fails.

If a database is created using an encryption key, the database file is strongly encrypted using either the AES 256-bit or AES FIPS-certified algorithm. By using strong encryption, you have increased security against skilled and determined attempts to gain access to the data. However, the use of strong encryption has a significant performance impact.
Any leading or trailing spaces in parameter values are ignored. The value cannot include leading single quotes, leading double quotes, or semicolons.

See also
- “UltraLite connection strings and parameters” on page 31
- “Database security” on page 26
- “UltraLite obfuscate creation parameter” on page 143
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

UltraLite DBN connection parameter
Differentiates databases by name when applications connect to more than one database.

Syntax

DBN=db-name

Default
None.

Behavior
1. On connect, look to see if the database is already running. If DBN is specified, look for a matching database and connect if found, proceed to auto-start if not.

2. If DBF is specified, look for a matching database (identical filename) and connect if found, proceed to auto-start if not.

3. If neither DBN nor DBF is specified, and a single database is running, connect to it.

4. A database is auto-started when required if DBF is specified. If DBN is also specified, it becomes the name of the running database, otherwise a name is generated from the base filename.

Remarks
UltraLite sets the database name after the database has been opened. Client applications can then connect to this database via its name instead of its file.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.
See also

- “UltraLite connection strings and parameters” on page 31
- “UltraLite DBF connection parameter” on page 165
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Example

Use the following parameters to connect to the running UltraLite database named Kitchener:

```
DBN=Kitchener;DBF=cities.udb
```

UltraLite desktop connection parameter

When creating a new UltraLite database, this connection parameter names the new database file.

When opening a connection to an existing database, it identifies the database.

Syntax

```
desktop:DBF=path\db | temp_dir=\Temp
```

Remarks

You should use the desktop connection parameter for UltraLite client applications that use the same connection string to connect to a Microsoft Windows or Mac device.

Colon (:) is considered to be a separator in addition to underscore (_). Options with a prefix take precedence over options without a prefix.

A `temp_dir` connection parameter is now available. This must name a directory (which already exists). UL will place the temporary file (with name still derived from the database name) in the specified directory, rather than beside the database file (the default and previous behavior). Specifying a temporary directory with faster I/O characteristics can improve the performance of things like temporary tables which are large relative to the cache size. Long-running transactions can also consume noticeable space in the temp file.

The value of desktop must meet the file name requirements for Windows or Mac. If you include an absolute path to the database, then all directories must exist before setting the path to this file. UltraLite does not create them automatically.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.
Behavior

1. If DBF is specified, look for a matching database (identical filename) and connect if found, proceed to auto-start if not.

2. A database is auto-started when required if DBF is specified.

See also

- “UltraLite connection strings and parameters” on page 31
- “UltraLite file path formats in connection parameters” on page 33
- “Precedence of connection parameters for UltraLite administration tools” on page 31
- “UltraLite DBF connection parameter” on page 165
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Example

The following example creates a new connection and identifies different database files for the Windows desktop and Windows Mobile platforms:

```
"desktop:DBF=C:\dir\db.udb; device:DBF=\SD Card\db.udb; device:temp_dir=\Temp; device:cache_size=4M"
```

UltraLite device connection parameter

When creating a new UltraLite database, this connection parameter names the new database file.

When opening a connection to an existing database, it identifies the database.

Syntax

```
device:DBF=path\db | temp_dir=\Temp
```

Remarks

You should use the device connection parameter for UltraLite client applications that use the same connection string to connect to a Microsoft Windows CE or iPhone device.

Colon (:) is considered to be a separator in addition to underscore (_). Options with a prefix take precedence over options without a prefix.

A **temp_dir** connection parameter is now available. This must name a directory (which already exists). UL will place the temporary file (with name still derived from the database name) in the specified directory, rather than beside the database file (the default and previous behavior). Specifying a temporary directory with faster I/O characteristics can improve the performance of things like temporary tables which are large relative to the cache size. Long-running transactions can also consume noticeable space in the temp file.
The value of device must meet the file name requirements for Windows CE or iPhone. If you include an absolute path to the database, then all directories must exist before setting the path to this file. UltraLite does not create them automatically.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

See also
- “UltraLite connection strings and parameters” on page 31
- “UltraLite file path formats in connection parameters” on page 33
- “Precedence of connection parameters for UltraLite administration tools” on page 31
- “UltraLite DBF connection parameter” on page 165
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite -.NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite -.NET Programming]

Example
The following example creates a new connection and identifies different database files for the Windows desktop and Windows Mobile platforms:

"desktop:DBF=C:\dir\db.udb; device:DBF=\SD Card\db.udb; device:temp_dir=\Temp; device:cache_size=4M"

UltraLite MIRROR_FILE connection parameter

Specifies the name of the database mirror file to which all database writes will be issued (at the same time as they are to the main database file).

Syntax

```
MIRROR_FILE=path\mirrorfile-db
```

Default
None.

Remarks
UltraLite provides basic database file mirroring to improve fault tolerance on potentially unreliable storage systems. This is accomplished using the mirror file. All database writes are issued to the mirror file at the same time as they are to the main database file (write overhead is therefore doubled; read overhead is not affected). If a corrupt page is read from the database file, the page is recovered by reading from the mirror file.

Mirroring is supported on all platforms using a file-based store.
When the `mirror_file=` option is specified when you start the database, UltraLite will open the named file and verify that it matches the main database file before continuing. If the mirror file does not exist, it is created at that point by copying the main file. If the mirror is not a database file, or is corrupt, an error is reported and the database will not start until the file is removed or a different mirror is specified. If the mirror does not match the database, `SQLE_MIRROR_FILE_MISMATCH` is generated and the database will not start. When a corrupt page is recovered, the warning `SQLE_CORRUPT_PAGE_READ_RETRY` is generated. (Without mirroring, or if the mirror file is also corrupt, the error `SQLEDEVICE_ERROR` is generated and the database is halted.)

To effectively protect against media failures, page checksums must be enabled when you use a mirror file. (With or without mirroring, page checksums allow UltraLite to detect page corruption as soon as the page is loaded and avoid referencing corrupt data.) Specify the `checksum_level` database creation option to enable checksums. UltraLite will generate the warning `SQLE_MIRROR_FILE_REQUIRES_CHECKSUMS` if checksums are not enabled when using a mirror file.

Because the mirror is an exact copy of the database file, it can be started directly as a database. The `ulvalid` utility will report corrupt pages.

**See also**

- “UltraLite checksum_level creation parameter” on page 134
- “UltraLite Validate Database utility (ulvalid)” on page 219
- “UltraLite connection strings and parameters” on page 31
- “UltraLite file path formats in connection parameters” on page 33
- “Precedence of connection parameters for UltraLite administration tools” on page 31
- UltraLite for C/C++: “UltraLite database connections” [*UltraLite - C and C++ Programming*]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [*UltraLite - C and C++ Programming*]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [*UltraLite - C and C++ Programming*]
- UltraLite.NET: “Connection setup for an UltraLite database” [*UltraLite - .NET Programming*]
- “ULConnection.Open method [UltraLite.NET]” [*UltraLite - .NET Programming*]

**Example**

The following example creates a new connection and creates a mirror file:

```c
Connection = DatabaseMgr.OpenConnection("DBF=c:\Dbfile.udb; UID=J Doe;PWD=ULdb; MIRROR_FILE=c:\test\MyMirrorDB.udb")
```

**UltraLite NT_FILE connection parameter**

When creating a new UltraLite database, this connection parameter names the new database file.

When opening a connection to an existing database, it identifies the database.

For Android smartphones, you can use `Configuration.setDatabaseName` as an equivalent to setting this connection parameter. See “`Configuration.setDatabaseName method [UltraLiteJ]`” [*UltraLite - Java Programming*].
Syntax

\texttt{NT\_FILE=\textbackslash path\nt-db}

Default

DBF connection parameter.

Remarks

You should use the NT\_FILE connection parameter for UltraLite client applications that use the same connection string to connect to a desktop database, and a database on other platforms.

This connection parameter takes precedence over the DBF parameter. If you are connecting from an UltraLite administration tool, or your connection object only connects to a desktop database, use the DBF connection parameter.

The value of NT\_FILE must meet the file name requirements for Windows desktop platforms.

The path can be absolute or relative. If you include a directory as part of the file name, then all directories must exist before setting the path to this file. UltraLite does not create them automatically.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

See also

- “UltraLite connection strings and parameters” on page 31
- “UltraLite file path formats in connection parameters” on page 33
- “Precedence of connection parameters for UltraLite administration tools” on page 31
- “UltraLite DBF connection parameter” on page 165
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Example

The following example creates a new connection and identifies different database files for the desktop and Windows Mobile platforms:

\begin{verbatim}
Connection = DatabaseMgr.OpenConnection("UID=JDoe;PWD=ULdb;
NT\_FILE=c:\test\MyTestDB.udb;CE\_FILE=\database\MyCEDB.udb")
\end{verbatim}

\textbf{UltraLite PWD connection parameter}

When creating a new UltraLite database, this connection parameter sets the password for the default user.

When connecting to an existing database, it defines the password for a user ID that is used for authentication.
For Android smartphones, you can use Configuration.setPassword as an alternative to setting this connection parameter. See “Configuration.setPassword method [UltraLiteJ]” [UltraLite - Java Programming].

**Syntax**

`PWD=password`

**Default**

If you do not set both the UID and PWD, UltraLite opens connections with `UID=DBA` and `PWD=sql`.

**Remarks**

Every user of a database has a password. UltraLite supports up to four user ID/password combinations.

You can set passwords to NULL or an empty string.

A random 4-byte salt value is generated when a new user is created or an existing user changes their password. The salt value is appended to the user's password when calculating the password hash and is stored in the database along with the hash. Salting significantly decreases vulnerability to dictionary attacks and also ensures that users with the same password will have different password hashes.

This connection parameter is not encrypted. However, UltraLite hashes the password before saving it, so you can only modify a password using Sybase Central.

**See also**

- “Users” [SQL Anywhere Server - Database Administration]
- “UltraLite connection strings and parameters” on page 31
- “UltraLite users” on page 56
- “UltraLite UID connection parameter” on page 176
- UltraLite for C/C++: “User authentication” [UltraLite - C and C++ Programming]
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “User authentication” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite -.NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite -.NET Programming]

**Examples**

The following partial connection string supplies the user ID DBA and password sql:

```
"UID=DBA;PWD=sql"
```

The following partial connection string supplies the user ID DBA and an empty password:

```
"UID=DBA;PWD=\"\"
```
**UltraLite RESERVE_SIZE connection parameter**

Pre-allocates the file system space required for your UltraLite database, without actually inserting any data. Reserving the file system space prevents the space from being used up by other files.

### Syntax

```
RESERVE_SIZE= number{ k | m | g }
```

### Default

0 (no reserve size).

### Remarks

The value you supply can be any value from 0 to your maximum database size. Use k, m, or g to specify units of kilobytes, megabytes, or gigabytes, respectively. If you do not specify a unit, bytes are assumed by default.

You should run the database with test data and observe the database size and choose a reserve size that suits your UltraLite deployment.

If the RESERVE_SIZE value is smaller than the database size, UltraLite ignores the parameter.

Reserving file system space can improve performance slightly because it may:

- Reduce the degree of file fragmentation compared to growing incrementally.
- Prevent out-of-storage memory failures.

Because an UltraLite database consists of data and metadata, the database size grows only when required (when the application updates the database).

### See also

- “UltraLite connection strings and parameters” on page 31
- “UltraLite CACHE_SIZE connection parameter” on page 160
- “UltraLite page_size creation parameter” on page 144
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite -.NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite -.NET Programming]

### Example

The following connection string fragment sets the reserve size to 128 KB so the system reserves that much system space for the database upon startup.

```
*RESERVE_SIZE=128K*
```
UltraLite START connection parameter

Starts the UltraLite engine executable. This parameter is not supported for UltraLite for Android. This parameter is only required if the engine is not in one of the expected locations.

Syntax

START=path\uleng16.exe

Remarks

Only supply a StartLine (START) connection parameter if you are connecting to an engine that is not currently running.

Paths with spaces require quotes. Otherwise, the client returns SQLE_UNABLE_TO_CONNECT_OR_START.

See also

- “UltraLite engine startup” on page 117
- “UltraLite Engine utility (uleng16)” on page 195
- “UltraLite data management components for Windows Mobile” on page 16
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]+” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Example

The following command starts the UltraLite engine that is located in the Program Files directory:

Start="Program Files\uleng16.exe"

An alternative way to define this path is to put the entire string in single quotes:

Start='"\Program Files\uleng16.exe"'

UltraLite TEMP_DIR connection parameter

 Specifies the name of the directory (which must already exist) into which UltraLite will place the temporary file (with a name derived from the database name).

Syntax

TEMP_DIR=path

Remarks

In addition to the database file, UltraLite creates and maintains a temporary file during database operation. You do not need to work with or maintain the file in any way.
By default, UltraLite maintains its temporary file in the same folder (if one exists) as the UltraLite database itself. The temporary file has the same file name as the database, but for file-based platforms the tilde is included in the extension of the file. For example, if you run the CustDB.udb sample database, the temporary file called CustDB.~db is maintained in the same directory as the database file.

Specifying a temporary directory with faster I/O characteristics can improve the performance of things like temporary tables which are large relative to the cache size. Long-running transactions can also consume noticeable space in the temp file.

Paths with spaces require quotes. Otherwise, the client returns SQL_E_UNABLE_TO_CONNECT_OR_START.

See also
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- ULDatabaseManager.OpenConnection method [UltraLite C++] [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Example
The following connection string fragment puts the temp file in the Temp directory:

```sql
  temp_dir=\Temp;
```

UltraLite UID connection parameter

When creating a new UltraLite database, this connection parameter sets the default user ID for the database.

When connecting to an existing database, it specifies the user ID with which you connect to the database. The value must be an authenticated user for the database.

Syntax

```sql
  UID=user
```

Default

If you do not set the UID and PWD when connecting, UltraLite opens connections with UID=DBA and PWD=sql.

Remarks

Every user of a database has a user ID. UltraLite supports up to four user ID/password combinations.

UltraLite user IDs are separate from MobiLink user names and from other SQL Anywhere user IDs. You cannot change a user ID once it is created. Instead, you must delete the user ID and then add a new one.
You cannot set the UID to NULL or an empty string. The maximum length for a user ID is 31 characters. User IDs are case insensitive.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes, leading double quotes, or semicolons.

See also

- “UltraLite connection strings and parameters” on page 31
- “UltraLite users” on page 56
- UltraLite for C/C++: “User authentication” [UltraLite - C and C++ Programming]
- UltraLite for C/C++: “UltraLite database connections” [UltraLite - C and C++ Programming]
- “ULDatabaseManager.OpenConnection method [UltraLite C++]” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “User authentication” [UltraLite - C and C++ Programming]
- UltraLite for embedded SQL: “UltraLite database connection using embedded SQL” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Connection setup for an UltraLite database” [UltraLite - .NET Programming]
- “ULConnection.Open method [UltraLite.NET]” [UltraLite - .NET Programming]

Example

The following connection string fragment supplies the user ID DBA and password sql for a database:

"UID=DBA;PWD=sql"

UltraLite database properties

UltraLite database property values are defined when the database is first created. They can be changed by re-creating the UltraLite database or editing their corresponding database option, if available.

UltraLite supports the following database properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache_allocation</td>
<td>Returns the current cache size as a percentage of the minimum and maximum settings. See “UltraLite cache_allocation option” on page 181.</td>
</tr>
<tr>
<td>CaseSensitive</td>
<td>Returns the status of the case sensitivity feature. Returns On if the database is case sensitive. Otherwise, it returns Off. See “UltraLite case creation parameter” on page 133.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CharSet</td>
<td>Returns the CHAR character set of the database. The character set used by the database is determined by the database's collation sequence and whether the data is UTF-8 encoded.</td>
</tr>
<tr>
<td></td>
<td>See also:</td>
</tr>
<tr>
<td></td>
<td>● “UltraLite utf8_encoding creation parameter” on page 155</td>
</tr>
<tr>
<td></td>
<td>● “UltraLite collation creation parameter” on page 135</td>
</tr>
<tr>
<td>ChecksumLevel</td>
<td>Returns the level of checksum validation in the database, one of 0 (do not add checksums), 1 (add checksums only to important pages), or 2 (add checksums to all pages). See “UltraLite checksum_level creation parameter” on page 134.</td>
</tr>
<tr>
<td>Collation</td>
<td>Returns the name of the database's collation sequence. See “UltraLite collation creation parameter” on page 135.</td>
</tr>
<tr>
<td>commit_flush_count</td>
<td>Returns the value of the commit_flush_count option that sets a commit count threshold. See “UltraLite commit_flush_count option [temporary]” on page 182.</td>
</tr>
<tr>
<td>commit_flush_timeout</td>
<td>Returns the value of the commit_flush_timeout option that sets a time interval threshold. See “UltraLite commit_flush_timeout option [temporary]” on page 183.</td>
</tr>
<tr>
<td>ConnCount</td>
<td>Returns the number of connections to the database. The value is dynamic: it can vary depending on how many connections currently exist. UltraLite supports up to fourteen concurrent database connections.</td>
</tr>
<tr>
<td>date_format</td>
<td>Returns the date format the database uses for string conversions. See “UltraLite date_format creation parameter” on page 136.</td>
</tr>
<tr>
<td>date_order</td>
<td>Returns the date order the database uses for string conversions. See “UltraLite date_order creation parameter” on page 138.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Encryption             | Returns the type of database encryption, one of None, Simple, AES, or AES_FIPS. The encryption used by the database is determined by whether or not you have configured strong encryption (AES or AES_FIPS) and the DBKEY creation parameter, or obfuscation (simple encryption). The only time this property can change is when the value is originally None (that is, neither fips nor obfuscation is used) and you then change the encryption key by specifying a new encryption key on the Connection object by calling the correct function or method for your API. In this case, the value would change to AES because the fips creation parameter cannot be set after the database has been created. See:  
  - “ULChangeEncryptionKey method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]  
  - “ULConnection.ChangeEncryptionKey method [UltraLite.NET]” [UltraLite -.NET Programming]  
  - “Database security” on page 26  
  - “UltraLite fips creation parameter” on page 140  
  - “UltraLite obfuscate creation parameter” on page 143  
  - “UltraLite DBKEY connection parameter” on page 166  |
| File                   | Returns the name of the database root file for the current connection, including the path. This is the value specified in the DBF connection parameter value. See:  
  - “UltraLite DBF connection parameter” on page 165  |
<p>| global_database_id     | Returns the value of the global_database_id option used for global autoincrement columns. See “UltraLite global_database_id option” on page 184.  |
| isolation_level        | Returns the current isolation level of the database. The value can either be read_committed or read_uncommitted. See “Isolation levels” on page 39.  |
| MaxHashSize            | Returns the default number of maximum bytes to use for index hashing. This property can be set on a per-index basis. See “UltraLite max_hash_size creation parameter” on page 141.  |
| ml_remote_id           | Returns the value of the ml_remote_id option that uniquely identifies the database for MobiLink synchronization. See “UltraLite ml_remote_id option” on page 185.  |</p>
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| Name                | Returns the name (or alias) of the database for the current connection. The name returned matches the DBN connection parameter value. If you did not use the DBN connection parameter, the name returned is the database file without the path and extension.  
See also:  
  - “UltraLite DBN connection parameter” on page 167  
  - “UltraLite DBF connection parameter” on page 165 |
| nearest_century     | Returns the nearest century the database uses for string conversions. See “UltraLite nearest_century creation parameter” on page 142.            |
| PageSize            | Returns the page size of the database, in bytes. See “UltraLite page_size creation parameter” on page 144.                                    |
| precision           | Returns the floating-point precision the database uses for string conversions. See “UltraLite precision creation parameter” on page 146.       |
| scale               | Returns the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum PRECISION during string conversions by the database. See “UltraLite scale creation parameter” on page 147. |
| time_format         | Returns the time format the database uses for string conversions. See “UltraLite time_format creation parameter” on page 148.                  |
| timestamp_format    | Returns the timestamp format the database uses for string conversions. See “UltraLite timestamp_format creation parameter” on page 150.          |
| timestamp_increment | Returns the minimum difference between two unique timestamps, in microseconds. See “UltraLite timestamp_increment creation parameter” on page 152. |
| timestamp_with_time_zone | Returns the timestamp format for TIMESTAMP WITH TIME ZONE values. See “UltraLite timestamp_with_time_zone_format creation parameter” on page 153. |
UltraLite database options

This section describes the UltraLite database options that are available.

UltraLite database option values are defined when the database is first created and can be altered while connected to the database.

See also
- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “UltraLite creation parameters” on page 131
- “UltraLite database options” on page 181

UltraLite cache_allocation option

Explicitly resizes the cache. The value is a percentage of the minimum-to-maximum range. A value of zero means the minimum size, and a value of 100 means the maximum size.

Allowed values
- Integer

Default
- None.

Remarks
- The cache_allocation property returns the current cache size as a percentage value of the minimum and maximum cache size.
UltraLite commit_flush_count option [temporary]

Sets a commit count threshold, after which a commit flush is performed.

Allowed values

Integer

Default

10

Remarks

Use 0 to disable the transaction count. When the transaction count is disabled, the number of commits is unlimited when a flush is triggered.

Both commit_flush_count and commit_flush_timeout are temporary database options. You must set these options each time you start a database. They persist as long as the database continues to run. They are only required when you set COMMIT_FLUSH=grouped as part of a connection string.

When you set this option and set the COMMIT_FLUSH connection parameter to grouped in your connection string, either threshold triggers a flush. When the flush occurs, UltraLite sets the counter and the timer back to 0. Then, both the counter and timer are monitored until one of these thresholds is subsequently reached.

An important consideration for setting the commit flush options is how much the delay to flush committed transactions poses a risk to the recoverability of your data. There is a slight chance that a transaction may be lost, even though it has been committed. If a serious hardware failure occurs after a commit, but before the transaction is flushed to storage, the transaction is rolled back on recovery. A longer delay can increase UltraLite performance. You must choose an appropriate count threshold with care.

To set the commit_flush_count option from a client application, set the option using the set database option function for the programming interface you are using or use the SET OPTION SQL statement.
UltraLite commit_flush_timeout option [temporary]

Sets a time interval threshold, after which a grouped commit flush is performed.

**Allowed values**

Integer, in milliseconds

**Default**

10000 milliseconds

**Remarks**

Use 0 to disable the time threshold.

Both commit_flush_count and commit_flush_timeout are temporary database options. You must set these options each time you start a database. They persist as long as the database continues to run. They are only required when you set COMMIT_FLUSH=grouped as part of a connection string.

If you set this option in addition to the commit_flush_timeout option and if you have set the COMMIT_FLUSH connection parameter to grouped, either threshold triggers a flush. When the flush occurs, UltraLite sets the counter and the timer back to 0. Then, both the counter and timer are monitored until one of these thresholds is subsequently reached.

An important consideration for setting the commit flush options is how much the delay to flush committed transactions poses a risk to the recoverability of your data. There is a slight chance that a transaction may be lost, even though it has been committed. If a serious hardware failure occurs after a commit, but before the transaction is flushed to storage, the transaction is rolled back on recovery. A longer delay can increase UltraLite performance. You must choose an appropriate timeout threshold with care.

To set the commit_flush_timeout option from a client application, set it using the set database option function for the programming interface you are using or use the SET OPTION SQL statement.
UltraLite database reference

See also

- “Accessing database options” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “UltraLite commit_flush_count option [temporary]” on page 182
- “UltraLite COMMIT_FLUSH connection parameter” on page 163
- “ULConnection.SetDatabaseOptionInt method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULSetDatabaseOptionULong method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULDatabaseSchema.SetDatabaseOption method [UltraLite.NET]” [UltraLite -.NET Programming]

UltraLite global_database_id option

Sets the database identification number.

Allowed values

Unique, non-negative integer

Default

The range of default values for a particular global autoincrement column is $pn + 1 \text{ to } p(n + 1)$, where $p$ is the partition size of the column and $n$ is the global database identification number.

Remarks

To maintain primary key uniqueness when synchronizing with a MobiLink server, the global ID sets a starting value for GLOBAL AUTOINCREMENT columns. The global ID must be set before default values can be assigned. If a row is added to a table and does not have a value set already, UltraLite generates a value for the column by combining the global_database_id value and the partition size.

When this option is set, UltraLite performs a commit.

When deploying an application, you must assign a different identification number to each database for synchronization with the MobiLink server. You can change the global ID of an existing database at any time.

To set the global_database_id option from a client application, use the set database option function for the programming interface you are using or use the SET OPTION SQL statement.
See also

- “GLOBAL AUTOINCREMENT” [MobiLink - Server Administration]
- “Accessing database options” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “GLOBAL AUTOINCREMENT columns in UltraLite” on page 64
- “ULConnection.SetDatabaseOptionInt method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULSetDatabaseID method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULConnection.DatabaseID property [UltraLite.NET]” [UltraLite - .NET Programming]
- “Connection.setDatabaseId method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_DATABASE_ID variable [BlackBerry] [UltraLiteJ]” [UltraLite - Java Programming]

Example

To autoincrement UltraLite database columns from 3001 to 4000, set the global ID to 3.

```
SET OPTION global_database_id=3
```

UltraLite isolation_level option

Isolation levels define the degree to which the operations in one transaction are visible to the operations in other concurrent transactions. UltraLite uses the default isolation level, read_committed, for connections in auto-commit mode.

See also

- “Accessing database options” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Isolation levels” on page 39
- “ULConnection.SetDatabaseOption method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “Data synchronization on a BlackBerry smartphone” [UltraLite - Java Programming]

UltraLite ml_remote_id option

The remote ID is a unique identifier for an UltraLite database that is used by MobiLink to identify the database for synchronization. The remote ID can be any string that has meaning to you, as long as the string remains unique among all remote MobiLink clients. The ID can also be set to NULL (NULL is the initial value). During synchronization, if the remote ID is NULL, UltraLite will assign it to a generated GUID.

If you prepopulate an UltraLite database using synchronization for distribution to multiple devices, you must reset the remote ID to NULL before distribution to ensure that each database has a unique remote ID. Upon distribution, a new unique remote ID can be set explicitly or it can be left as NULL so that UltraLite will automatically generate a new unique value.
Allowed values

Any value that uniquely identifies the database for MobiLink synchronization.

Default

Null

Remarks

MobiLink uses the remote ID to store the synchronization information for the remote database. Given the remote ID, MobiLink user names are no longer required to be unique. The remote ID becomes particularly useful when you have multiple MobiLink users synchronizing the same UltraLite database. In this case, your synchronization scripts should reference the remote ID and not just the user name.

When this option is set, UltraLite performs a commit.

To set the ml_remote_id option from a client application, set it using the set database option function for the programming interface you are using or use the SET OPTION SQL statement.

See also

- “Accessing database options” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Remote IDs” [MobiLink - Client Administration]
- “User Name synchronization parameter” on page 103
- “ULConnection.SetDatabaseOption method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULSetDatabaseOptionString method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_ML_REMOTE_ID variable [BlackBerry] [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite utilities

UltraLite includes a set of utilities that are designed to perform basic database administration activities at a command prompt. Many of these utilities share a similar functionality to the SQL Anywhere Server utilities. However, the way options are used can vary. Always refer to the UltraLite reference documentation for the UltraLite implementation of these options.

Note

Options for the utilities documented in this section are case sensitive, unless otherwise noted. Type options exactly as they are displayed.
Supported exit codes

The ulload, ulsync, and ulunload utilities return exit codes to indicate whether the operation a utility attempted to complete was successful. 0 indicates a successful operation. Any other value indicates that the operation failed.

<table>
<thead>
<tr>
<th>Exit code</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>EXIT_OKAY</td>
<td>Operation successful.</td>
</tr>
<tr>
<td>1</td>
<td>EXIT_FAIL</td>
<td>Operation failure.</td>
</tr>
<tr>
<td>3</td>
<td>EXIT_FILE_ERROR</td>
<td>Database cannot be found.</td>
</tr>
<tr>
<td>4</td>
<td>EXIT_OUT_OF_MEMORY</td>
<td>Exhausted the dynamic memory of the device.</td>
</tr>
<tr>
<td>6</td>
<td>EXIT_COMMUNICATIONS_FAIL</td>
<td>Communications error generated while talking to the UltraLite engine.</td>
</tr>
<tr>
<td>9</td>
<td>EXIT_UNABLE_TO_CONNECT</td>
<td>Invalid UID or PWD provided, therefore cannot connect to the database.</td>
</tr>
<tr>
<td>12</td>
<td>EXIT_BAD_ENCRYPT_KEY</td>
<td>Missing or invalid encryption key.</td>
</tr>
<tr>
<td>13</td>
<td>EXIT_DB_VER_NEWER</td>
<td>Detected that the database version is incompatible. The database must be upgraded to a newer version.</td>
</tr>
<tr>
<td>255</td>
<td>EXIT_USAGE</td>
<td>Invalid command line options.</td>
</tr>
</tbody>
</table>

Interactive SQL for UltraLite utility (dbisql)

Executes SQL statements and runs script files against a database.

Syntax

`dbisql -c "connection-string" [ options ] [ dbisql-statement | dbisql-script-file ]`

`dbisql -c "connection-string" -ul [ options ] [ dbisql-statement | dbisql-script-file ]`

`dbisql-statement`: A SQL statement or a series of sql statements separated by a command-delimiter.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Reads in options from the specified environment variable or configuration file.</td>
</tr>
<tr>
<td></td>
<td>If both the environment variable and configuration file exist with the same name, the environment variable is used. See “Configuration files” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td></td>
<td>To protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file. See “File Hiding utility (dbfhide)” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>-c &quot;keyword=value; ...&quot;</td>
<td>Specifies connection parameters. If Interactive SQL cannot connect, you are presented with a window where you can enter the connection parameters. If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed. See “Connection parameters” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>-d delimiter</td>
<td>Specifies a command delimiter. Quotation marks around the delimiter are optional, but are required when the command shell itself interprets the delimiter in some special way.</td>
</tr>
<tr>
<td></td>
<td>This option overrides the setting of the command_delimiter option. See “command_delimiter option [Interactive SQL]” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>-d1</td>
<td>Echoes all statements explicitly executed by the user to the command window (STDOUT). This can provide useful feedback for debugging SQL scripts, or when Interactive SQL is processing a long SQL script. (The final character is a number 1, not a lowercase L). This option is only available when you run Interactive SQL as a command line program.</td>
</tr>
<tr>
<td>-datasource DSN-name</td>
<td>Specifies an ODBC data source to connect to.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-f filename</code></td>
<td>Opens (but does not run) the file called <em>filename</em> in the SQL Statements pane. If the -f option is given, the -c option is ignored; that is, no connection is made to the database. The file name can be enclosed in quotation marks, and <em>must</em> be enclosed in quotation marks if the file name contains a space. If the file does not exist, or if it is really a directory instead of a file, Interactive SQL prints an error message and then quits. If the file name does not include a full drive and path specification, it is assumed to be relative to the current directory. This option is only supported when Interactive SQL is run as a windowed application.</td>
</tr>
<tr>
<td><code>-host hostname</code></td>
<td>Specifies the <em>hostname</em> or IP address of the computer on which the database server is running. You can use the name localhost to represent the current computer.</td>
</tr>
<tr>
<td><code>-nogui</code></td>
<td>Runs Interactive SQL as a console application, with no windowed user interface. This is useful for batch operations. If you specify either <em>dbisql-statement</em> or <em>dbisql-script-file</em>, then -nogui is assumed. In this mode, Interactive SQL sets the program exit code to indicate success or failure. On Windows operating systems, the environment variable ERRORLEVEL is set to the program exit code. See “Software component exit codes” [<em>SQL Anywhere Server - Programming</em>].</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>`-onerror { continue</td>
<td>exit }`</td>
</tr>
<tr>
<td></td>
<td>● <strong>Continue</strong> The error is ignored and Interactive SQL continues executing statements.</td>
</tr>
<tr>
<td></td>
<td>● <strong>Exit</strong> Interactive SQL terminates.</td>
</tr>
<tr>
<td><code>-q</code></td>
<td>Suppresses output messages. Sets the utility to run in quiet mode. This is useful only if you start Interactive SQL with a statement or script file. Specifying this option does not suppress error messages, but it does suppress the following:</td>
</tr>
<tr>
<td></td>
<td>● warnings and other non-fatal messages</td>
</tr>
<tr>
<td></td>
<td>● the printing of result sets</td>
</tr>
<tr>
<td><code>-ul</code></td>
<td>Specifies that UltraLite databases are the default. Interactive SQL customizes the options available to you depending on the type of database you are connected to. By default, Interactive SQL assumes that you are connecting to SQL Anywhere databases. When you specify the -ul option, the default changes to UltraLite databases. Regardless of the type of database set as the default, you can connect to either SQL Anywhere or UltraLite databases by choosing the database type from the Change Database Type dropdown list on the Connect window. For more information about connecting to UltraLite databases from Interactive SQL, see “Interactive SQL for UltraLite utility (dbisql)” on page 187.</td>
</tr>
</tbody>
</table>
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-version</td>
<td>Displays the version number of Interactive SQL. You can also view the version number from within Interactive SQL; from the <strong>Help</strong> menu, click <strong>About Interactive SQL</strong>.</td>
</tr>
</tbody>
</table>
| -x | Scans statements but does not execute them. This is useful for checking long script files for syntax errors.  
For detailed descriptions of SQL statements, see “**SQL statements**” [*SQL Anywhere Server - SQL Reference*]. |
| dbisql-statement | Execute the SQL statement or execute the specified `dbisql-script-file`.  
If you do not specify a `dbisql-statement` or `dbisql-script-file`, Interactive SQL enters interactive mode, where you can type a statement into a command window. |

### Remarks

Interactive SQL allows you to browse the database, execute SQL statements, and run script files. It also provides feedback about:

- the number of rows affected
- the time required for each statement
- the execution plan of queries
- any error messages

You can use Interactive SQL to connect to a SQL Anywhere database, an UltraLite database, an SAP Sybase IQ database, an SAP HANA database, or a generic ODBC database.

For Windows, there are two executables:

1. Batch scripts should call `dbisql` or `dbisql.com`, not `dbisql.exe`. The `dbisql.com` executable is linked as a console application.

2. The `dbisql.exe` executable is linked as a windowed application and does not block the command shell from which it was started. If `dbisql.exe` is run from a batch file, you won’t see any output sent to the standard output or standard error files.

The default encoding for Interactive SQL can also be temporarily set using the `default_isql_encoding` option. See “**default_isql_encoding option [Interactive SQL]**” [*SQL Anywhere Server - Database Administration*].
You can specify the encoding to use when reading or writing files using the ENCODING clause of the INPUT, OUTPUT, or READ statement. See:

- “INPUT statement [Interactive SQL]” [SQL Anywhere Server - SQL Reference]
- “OUTPUT statement [Interactive SQL]” [SQL Anywhere Server - SQL Reference]
- “READ statement [Interactive SQL]” [SQL Anywhere Server - SQL Reference]

Exit codes are 0 (success) or non-zero (failure). Non-zero exit codes are set only when you run Interactive SQL in batch mode (with a command line that contains a SQL statement or the name of a script file). See “Software component exit codes” [SQL Anywhere Server - Programming].

In command-prompt mode, Interactive SQL sets the program exit code to indicate success or failure. On Windows operating systems, the environment variable ERRORLEVEL is set to the program exit code.

When executing a reload.sql file with Interactive SQL and the database is encrypted, 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.

You can start Interactive SQL in the following ways:

- From Sybase Central, by clicking File » Open Interactive SQL.
- From the Start menu by clicking Start » Programs » SQL Anywhere 16 » Administration Tools » Interactive SQL.
- Using the dbisql command at a command prompt.

See also

- “SQL statements for Interactive SQL” [SQL Anywhere Server - Database Administration]
- “Interactive SQL” [SQL Anywhere Server - Database Administration]
- “Configuration files” [SQL Anywhere Server - Database Administration]
- “File Hiding utility (dbfhide)” [SQL Anywhere Server - Database Administration]
- “INPUT statement [Interactive SQL]” [SQL Anywhere Server - SQL Reference]
- “OUTPUT statement [Interactive SQL]” [SQL Anywhere Server - SQL Reference]
- “READ statement [Interactive SQL]” [SQL Anywhere Server - SQL Reference]
- “UltraLite connection parameters” on page 156
- “Supported exit codes” on page 187

Example

The following command runs the script file mycom.sql against the CustDB.udb database for UltraLite. Because a user ID and password are not defined, the default user ID DBA and password sql are assumed. The -onerror option is defined as Exit; so, if there is an error in the script file, the process terminates.

```
dbisql -ul -c DBF=CustDB.udb -onerror exit mycom.sql
```

SQL preprocessor for UltraLite utility (sqlpp)

Preprocesses a C/C++ program that contains embedded SQL (ESQL), so that code required for that program can be generated before you run the compiler. The table below describes the entire set of options for completeness, but the only relevant options for UltraLite are -eu and -wu.
Syntax

```sqlpp -u [ options ] esql-filename [ output-filename ]```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-d</code></td>
<td>Generate code that reduces data space size, but increases code size. Data structures are reused and initialized at execution time before use.</td>
</tr>
<tr>
<td><code>-e flag</code></td>
<td>This option flags as an error any static embedded SQL that is not part of a specified standard. The <code>level</code> value indicates the standard to use. For example, <code>sqlpp -e c03</code> flags any syntax that is not part of the core SQL/2003 standard.</td>
</tr>
<tr>
<td></td>
<td>The allowed values of <code>level</code> are:</td>
</tr>
<tr>
<td></td>
<td>● <code>c03</code> Flag syntax that is not core SQL/2003 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>p03</code> Flag syntax that is not full SQL/2003 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>c99</code> Flag syntax that is not core SQL/1999 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>p99</code> Flag syntax that is not full SQL/1999 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>e92</code> Flag syntax that is not entry-level SQL/1992 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>i92</code> Flag syntax that is not intermediate-level SQL/1992 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>f92</code> Flag syntax that is not full-SQL/1992 syntax</td>
</tr>
<tr>
<td></td>
<td>● <code>t</code> Flag non-standard host variable types</td>
</tr>
<tr>
<td></td>
<td>● <code>u</code> Flag syntax that is not supported by UltraLite</td>
</tr>
<tr>
<td></td>
<td>For compatibility with previous SQL Anywhere versions, you can also specify <code>e</code>, <code>I</code>, and <code>f</code>, which correspond to <code>e92</code>, <code>i92</code>, and <code>f92</code>, respectively.</td>
</tr>
<tr>
<td><code>-h width</code></td>
<td>Limits the maximum length of split lines output by <code>sqlpp</code> to <code>width</code> in the <code>.c</code> file. Backslash characters are added to the end of split lines, so that a C compiler can parse the split lines as one continuous line. The default value is no maximum line length (output lines are not split by default).</td>
</tr>
<tr>
<td><code>-k</code></td>
<td>Notify the preprocessor that the program to be compiled includes a user declaration of SQLCODE.</td>
</tr>
<tr>
<td><code>-m mode</code></td>
<td>Cursor updatability mode. Either HISTORICAL or READONLY.</td>
</tr>
<tr>
<td><code>-n</code></td>
<td>Generate line number information in the C file by using <code>#line</code> directives in the appropriate places in the generated code. Use this option to report source errors and to debug source on line numbers in the <code>esql-filename</code> file, rather than in the <code>output-filename</code> file.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-o O/S spec</td>
<td>Not applicable to UltraLite.</td>
</tr>
<tr>
<td>-q</td>
<td>Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.</td>
</tr>
<tr>
<td>-r-</td>
<td>Not applicable to UltraLite.</td>
</tr>
<tr>
<td>-s string-length</td>
<td>Set the maximum size string that the preprocessor will put into the C file. Strings longer than this value are initialized using a list of characters (‘a’, ‘b’, ‘c’, and so on). Most C compilers have a limit on the size of string literal they can handle. This option is used to set that upper limit. The default value is 500.</td>
</tr>
<tr>
<td>-u</td>
<td>Required for UltraLite. Generate output specifically required for UltraLite databases.</td>
</tr>
</tbody>
</table>
| -w level        | Flag non-conforming SQL syntax as a warning. The level value indicates the standard to use. For example, sqlpp -w c03 ... flags any SQL syntax that is not part of the core SQL/2003 syntax.  
|                 | The allowed values of level are:  
|                 |   ● c03  Flag syntax that is not core SQL/2003 syntax  
|                 |   ● p03  Flag syntax that is not full SQL/2003 syntax  
|                 |   ● c99  Flag syntax that is not core SQL/1999 syntax  
|                 |   ● p99  Flag syntax that is not full SQL/1999 syntax  
|                 |   ● e92  Flag syntax that is not entry-level SQL/1992 syntax  
|                 |   ● i92  Flag syntax that is not intermediate-level SQL/1992 syntax  
|                 |   ● f92  Flag syntax that is not full-SQL/1992 syntax  
|                 |   ● t    Flag non-standard host variable types  
|                 |   ● u    Flag syntax that is not supported by UltraLite  
|                 | For compatibility with previous SQL Anywhere versions, you can also specify e, I, and f, which correspond to e92, i92, and f92, respectively. |
| -x              | Change multibyte strings to escape sequences, so that they can be passed through a compiler.                                                |
| -z collation-sequence | Specify the collation sequence.                                                                                      |
Remarks
This preprocessor translates the SQL statements in the input-file into C/C++. It writes the result to the output-filename. The normal extension for source files containing embedded SQL is sqc. The default output-filename is the esql-filename base name with an extension of .c. However, if the esql-filename already has the .c extension, the default output extension is .cc.

The collation sequence is used to help the preprocessor understand the characters used in the source code of the program. For example, in identifying alphabetic characters suitable for use in identifiers. In UltraLite, collations include a code page plus a sort order. If you do not specify -z, the preprocessor attempts to determine a reasonable collation to use based on the operating system.

To see a list of supported collations (and their corresponding codepages), run the command ulinit -Z.

Tip
The SQL preprocessor (sqlpp) has the ability to flag static SQL statements in an embedded SQL application at compile time. This feature can be especially useful when developing an UltraLite application, to verify SQL statements for UltraLite compatibility. You can test compatibility of SQL for both SQL Anywhere and UltraLite applications by using either -e and/or -w options. For an overview of the SQL Flagger, see “SQL compliance testing using the SQL Flagger” [SQL Anywhere Server - SQL Usage].

See also
● “Embedded SQL” [SQL Anywhere Server - Programming]
● “UltraLite character sets” on page 24

Example
The following command preprocesses the srcfile.sqc embedded SQL file in quiet mode for an UltraLite application.

    sqlpp -u -q MyEsqlFile.sqc

UltraLite Engine utility (uleng16)
Manages concurrent UltraLite database connections from applications, and allows the UltraLite engine to run as a daemon using the -ud option.

Syntax
uleng16 [ -ud ] [ db-file-name ]

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ud db-file-name</td>
<td>Lets you run the engine so that it continues running after the current user session ends. When you start the daemon directly using the -ud option, the uleng16 command creates the daemon process and returns immediately (exiting and allowing the next command to be executed) before the daemon initializes itself or attempts to open any of the databases specified in the command.</td>
</tr>
</tbody>
</table>
Remarks
The UltraLite engine does not display a messages window on startup.

The UltraLite engine should be used by an application in scenarios where multiple processes could be accessing the same database at the same time. The engine is installed in the SQL Anywhere bin32 or bin64 directory because the UltraLite desktop administration tools use the engine to connect to databases.

Using the -ud option, you can run the UltraLite engine so that when you log off the computer, the database engine remains running. (Normally when you log off the computer, all applications associated with the session shut down.)

See also
● “UltraLite deployment” on page 107
● “How to build and deploy UltraLite C++ applications” [UltraLite - C and C++ Programming]
● “UltraLite data management components for Windows Mobile” on page 16
● “UltraLite Engine Stop utility (ulstop)” on page 196
● “UltraLite START connection parameter” on page 175

UltraLite Engine Stop utility (ulstop)
Stops the UltraLite engine.

Syntax
ulstop

Remarks
Use ulstop during development to shut down the engine manually. You typically do not require ulstop in live deployments.

See also
● “UltraLite data management components for Windows Mobile” on page 16
● “UltraLite Engine utility (uleng16)” on page 195

UltraLite Erase utility (ulerase)
Erases an UltraLite database.

Syntax
ulerase [ options ] [ db-file-name ]

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-k key</td>
<td>Specify the encryption key for an encrypted database.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--ek=key</td>
<td></td>
</tr>
<tr>
<td>-p</td>
<td>Specify that you want to be prompted for the encryption key.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--ep</td>
<td></td>
</tr>
<tr>
<td>--log</td>
<td>Log operations to the specified file.</td>
</tr>
<tr>
<td>-q</td>
<td>Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--quiet</td>
<td></td>
</tr>
<tr>
<td>-u uid,pwd</td>
<td>Specify the userid and password required to access the database.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--dba=uid,pwd</td>
<td></td>
</tr>
<tr>
<td>-?</td>
<td>Displays utility usage information and exits.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
<tr>
<td>db-file-name</td>
<td>Erase the specified database.</td>
</tr>
</tbody>
</table>

**Remarks**

The database must be accessible. The user ID and password combination must allow a connection, otherwise the database is not erased.

Encrypted databases require a key provided in the connection string, or using one of `-k key` or `-p`.

**See also**

- “Configuration files” [SQL Anywhere Server - Database Administration]

**UltraLite Information utility (ulinfo)**

Displays information about an UltraLite database.
Syntax
ulinfo -c options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.</td>
</tr>
<tr>
<td>-c &quot;connection-string&quot;</td>
<td>Supply database connection parameters. Required.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--connect=&quot;connection-string&quot;</td>
<td></td>
</tr>
<tr>
<td>-q</td>
<td>Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--quiet</td>
<td></td>
</tr>
<tr>
<td>--log=filename</td>
<td>Log operations to the specified file.</td>
</tr>
<tr>
<td>-?</td>
<td>Displays utility usage information and exits.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
</tbody>
</table>

Remarks
Warning messages generated when opening an UltraLite database are always displayed unless you use the -q option.

See also

- “Configuration files” [SQL Anywhere Server - Database Administration]
- “UltraLite connection parameters” on page 156
- “UltraLite global_database_id option” on page 184
- “UltraLite ml_remote_id option” on page 185

Example
Show basic database internals for a file named cv_dbattr.udb that has already been synchronized:

ulinfo -c DBF=cv_dbattr.udb

UltraLite Initialize Database utility (ulinit)

[This topic has been updated for build 1823.]
Creates a new UltraLite database.

This utility functions under one of the following modes:

- **Empty mode**  Creates an empty database with characteristics specified with the command line arguments.

- **Extract mode**  Creates a database based on a SQL Anywhere database.

  An initial schema is created that matches tables and indexes in the SQL Anywhere reference database. Many of the reference database characteristics are extracted and used in the new UltraLite database.

**Syntax**

```
ulinit options dbname
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used. See “Configuration files” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>-a &quot;keyword=value;...&quot;</td>
<td>Sets the utility to <strong>extract mode</strong> and connects to an existing database using the specified connection parameters.</td>
</tr>
<tr>
<td>OR</td>
<td>If this option is not present, the utility creates a new database using the specified connection parameters (<strong>empty mode</strong>).</td>
</tr>
<tr>
<td>--SAconnect=&quot;keyword=value;...&quot;</td>
<td></td>
</tr>
<tr>
<td>-c</td>
<td>Empty mode.</td>
</tr>
<tr>
<td>OR</td>
<td>Enforce case sensitivity on all string comparisons.</td>
</tr>
<tr>
<td>--case</td>
<td></td>
</tr>
<tr>
<td>-d</td>
<td>Extract mode.</td>
</tr>
<tr>
<td>OR</td>
<td>For each table in the new UltraLite database, copy data from the corresponding table in the SQL Anywhere database. The new database is initially empty unless you use this option.</td>
</tr>
<tr>
<td>--datacopy</td>
<td>By default, this data is not be uploaded in subsequent synchronizations. To include the data in the next upload synchronization, use -i with -d.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>--date_format=format</strong></td>
<td>Empty mode. Sets the format for dates retrieved from the database. See “UltraLite date_format creation parameter” on page 136.</td>
</tr>
<tr>
<td><strong>--date_order=date-format-interpretation</strong></td>
<td>Empty mode. Sets the interpretation of the date format. See “UltraLite date_order creation parameter” on page 138.</td>
</tr>
<tr>
<td><strong>-e value</strong></td>
<td>Empty mode. OR On or off, 1 or 0, and so on. This option controls the use of AES FIPS-certified encryption. See “Database security” on page 26 and “UltraLite fips creation parameter” on page 140.</td>
</tr>
<tr>
<td><strong>-f</strong></td>
<td>Extract mode. OR Fail if exact schema is not supported in UltraLite; otherwise, warnings will appear if schema differs.</td>
</tr>
<tr>
<td><strong>-g id</strong></td>
<td>Set the initial database ID to the INTEGER value you assign. This initial value is used with a partition size for new rows that have global autoincrement columns. When deploying an application, you must assign a different range of identification numbers to each database for synchronization with the MobiLink server. See “UltraLite global_database_id option” on page 184.</td>
</tr>
<tr>
<td><strong>--databaseid=id</strong></td>
<td></td>
</tr>
<tr>
<td><strong>--insertforupload</strong></td>
<td>Extract mode. OR Use with -d. Include inserted rows in the next upload synchronization. By default, rows inserted by this utility are not uploaded during synchronization.</td>
</tr>
<tr>
<td><strong>--identity-file=file</strong></td>
<td>Specify the file containing the client TLS identity. See “identity” [MobiLink - Client Administration].</td>
</tr>
<tr>
<td><strong>--identity-password=password</strong></td>
<td>Specify the password for the client TLS identity. See “identity_password” [MobiLink - Client Administration].</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-k key</code></td>
<td>Extract mode. Specify the encryption key for an encrypted database.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td><code>--key=key</code></td>
<td></td>
</tr>
<tr>
<td><code>-K</code></td>
<td>Empty mode. Specify that you want to be prompted for the encryption key.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td><code>--prompt</code></td>
<td></td>
</tr>
<tr>
<td><code>-l filename</code></td>
<td>Extract mode. Log DDL database schema creation SQL statements, as executed, to the specified file.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td><code>--sql=filename</code></td>
<td></td>
</tr>
<tr>
<td><code>--log=filename</code></td>
<td>Empty mode. Log operations to the specified file.</td>
</tr>
<tr>
<td><code>-m filename</code></td>
<td>Extract mode. Specify the database mirror file. See “UltraLite MIRROR_FILE connection parameter” on page 170.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td><code>--mirror_file=filename</code></td>
<td></td>
</tr>
<tr>
<td><code>--max_hash_size=size</code></td>
<td>Empty mode. Sets the default index hash size in bytes. See “UltraLite max_hash_size creation parameter” on page 141.</td>
</tr>
<tr>
<td><code>-n pubname</code></td>
<td>Extract mode. Required. Add tables to the UltraLite database schema.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td><code>--publication=pubname</code></td>
<td></td>
</tr>
</tbody>
</table>

*pubname* specifies a publication in the reference database. Tables in the publication are added to the UltraLite database. Specify the option multiple times to add tables from multiple publications to the UltraLite database. To add all tables in the reference database to the UltraLite database, specify `-n*`. 
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--nearest_century=yy</td>
<td>Empty mode. Controls the interpretation of two-digit years in string-to-date conversions. See “UltraLite nearest_century creation parameter” on page 142.</td>
</tr>
<tr>
<td>-o value</td>
<td>Empty mode. On or off, 1 or 0, and so on. Controls whether data in the database is obfuscated. Obfuscation is a form of simple encryption. See “Database security” on page 26 and “UltraLite obfuscate creation parameter” on page 143.</td>
</tr>
<tr>
<td>-p size</td>
<td>Empty mode. Specify the database page size.</td>
</tr>
<tr>
<td>--page_size=size</td>
<td>Empty mode. Specify the database page size.</td>
</tr>
<tr>
<td>--precision=precision</td>
<td>Empty mode. Specifies the maximum number of digits in decimal point arithmetic results. See “UltraLite precision creation parameter” on page 146.</td>
</tr>
<tr>
<td>-q</td>
<td>Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages and warnings are still displayed, however.</td>
</tr>
<tr>
<td>--quiet</td>
<td>Empty mode. Specify the database page size.</td>
</tr>
<tr>
<td>-r size</td>
<td>Database connection only. Reserve size. See “UltraLite RESERVE_SIZE connection parameter” on page 174.</td>
</tr>
<tr>
<td>--reserve_size</td>
<td>Empty mode. Specify the database page size.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-s pubname</code></td>
<td>Extract mode. Create a publication in the UltraLite database with the same definition as <code>pubname</code> in the reference database. Publications are used to configure synchronization. Supply more than one <code>-s</code> option to name more than one synchronization publication. The tables in this publication must be included in a publication listed by the <code>-n</code> option. If <code>-s</code> is not supplied, the UltraLite remote database has no named publications. For more information about how to create publications for MobiLink synchronization, see “Publishing data in UltraLite” on page 72.</td>
</tr>
<tr>
<td><code>-S checksum_level</code></td>
<td>Empty mode. 0, 1, or 2. Specifies the checksum level validation on database pages. See “UltraLite checksum_level creation parameter” on page 134.</td>
</tr>
<tr>
<td><code>--sync_publication</code></td>
<td>Empty mode. Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision. See “UltraLite scale creation parameter” on page 147.</td>
</tr>
<tr>
<td><code>--checksum_level=checksum_level</code></td>
<td>Empty mode. Specify the file containing the trusted root certificate. This certificate is required for server authentication.</td>
</tr>
<tr>
<td><code>--scale=scale</code></td>
<td>Empty mode. Sets the format for times retrieved from the database. See “UltraLite time_format creation parameter” on page 148.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--timestamp_format=format</td>
<td>Empty mode.</td>
</tr>
<tr>
<td></td>
<td>Sets the format for timestamps retrieved from the database. See “UltraLite timestamp_format creation parameter” on page 150.</td>
</tr>
<tr>
<td>--timestamp_increment=increment</td>
<td>Empty mode.</td>
</tr>
<tr>
<td></td>
<td>Determines how the timestamp is truncated in UltraLite. See “UltraLite timestamp_increment creation parameter” on page 152.</td>
</tr>
<tr>
<td>--timestamp_with_time_zone_format=format</td>
<td>Empty mode.</td>
</tr>
<tr>
<td></td>
<td>This option sets the format for TIMESTAMP WITH TIME ZONE values retrieved from the database. See “UltraLite timestamp_with_time_zone_format creation parameter” on page 153.</td>
</tr>
<tr>
<td>-u &lt;uid&gt;,&lt;pwd&gt;</td>
<td>Database connection only.</td>
</tr>
<tr>
<td>OR</td>
<td>Specify the userid and password.</td>
</tr>
<tr>
<td>--dba=&lt;uid&gt;,&lt;pwd&gt;</td>
<td></td>
</tr>
<tr>
<td>--utf8_encoding=value</td>
<td>Empty mode.</td>
</tr>
<tr>
<td></td>
<td>On or off, 1 or 0, and so on. Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode. See “UltraLite character sets” on page 24 and “UltraLite utf8_encoding creation parameter” on page 155.</td>
</tr>
<tr>
<td>-w</td>
<td>Extract mode.</td>
</tr>
<tr>
<td>OR</td>
<td>Do not display warnings.</td>
</tr>
<tr>
<td>--nowarnings</td>
<td></td>
</tr>
<tr>
<td>-x table</td>
<td>Extract mode.</td>
</tr>
<tr>
<td>OR</td>
<td>Exclude the tables named in the list.</td>
</tr>
<tr>
<td>--exclude</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>-y OR --overwrite</td>
<td>Over-write the existing database file.</td>
</tr>
<tr>
<td>-z OR --collation=collation-sequence</td>
<td>Empty mode.</td>
</tr>
<tr>
<td>-Z OR --listcollation</td>
<td>Empty mode. List the available collation sequences and exit.</td>
</tr>
<tr>
<td>-? OR --help</td>
<td>Display utility usage and exit.</td>
</tr>
</tbody>
</table>

**Note**
An option can be used in either mode if one is not specified in its description.

**Remarks**
When run in extract mode, ULINIT attempts to create an UltraLite database that matches, as closely as possible, the SQL Anywhere database. For example, if a column in the SQL Anywhere database includes a clause that UltraLite does not support, the default value is ignored and the UltraLite default used instead. A warning is generated and creation continues. This supports the case where SQL Anywhere tables cannot be modified, but a reasonable UltraLite alternative is available. To enforce an exact schema match, use the -f option. The ulinit utility fails if the schema does not support a reasonable UltraLite alternative.

**See also**
- “Conversion from a SQL Anywhere database to an UltraLite database” on page 29
- “Synchronization models” [MobiLink - Getting Started]
- “UltraLite connection parameters” on page 156

**Examples**
Create a file called customer.udb that contains the tables defined in TestPublication:

```
ulinit -a "DSN=MySADb;UID=JimmyB;PWD=secret" -n TestPublication -k mykey
customer.udb
```
This example connects to a SQL Anywhere database defined in the MySADb datasource. It creates an UltraLite database with all the database options from that database and all the tables contained in the TestPublication publication. The new UltraLite database is called customer.udb and is encrypted with the key mykey.

Create a file called customer.udb that contains two distinct publications. Specifically, Pub1 may contain a small subset of data for priority synchronization, while Pub2 could contain the bulk of the data:

```
ulinit -a "DSN=MySADb;UID=JimmyB;PWD=secret" --exactschema -n Pub1 -n Pub2 -s Pub1 -s Pub2 customer.udb
```

This example connects to a SQL Anywhere database defined in the MySADb datasource. It creates an UltraLite database with all the database options from that database and all the tables contained in the publications Pub1 and Pub2. The new UltraLite database is also created with the publications Pub1 and Pub2. Since the --exactschema option is set, ulinit will fail if it cannot extract the all precise schema.

Create a new blank database that overwrites another customer.udb file if it already exists. The new database has no schema and all the database options are set to default values.

```
ulinit -y customer.udb
```

## UltraLite Load XML to Database utility (ulload)

[This topic has been updated for build 1823.]

Loads data from an XML file into a new or existing database.

### Syntax

```
uload -c "connection-string" [ options ] xml-file
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used. See “Configuration files” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>-a</td>
<td>Add data and schema definitions into an existing database.</td>
</tr>
<tr>
<td>OR</td>
<td>--append</td>
</tr>
<tr>
<td>-c &quot;connection-string&quot;</td>
<td>Supply the database connection parameters.</td>
</tr>
<tr>
<td>OR</td>
<td>--connect=&quot;connection-string&quot;</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-d</td>
<td>Load data only, ignoring any schema metadata in the XML file input. -d or --dataonly switches can only be used when -a is specified (because it is loading data only, the UDB it is loading the data into must exist with a schema that supports the data being loaded into it).</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--dataonly</td>
<td></td>
</tr>
<tr>
<td>-e value</td>
<td>Specify on or off, 1 or 0, and so on. This option controls the use of AES FIPS-certified encryption. See “Database security” on page 26 and “UltraLite fips creation parameter” on page 140.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--fips value</td>
<td></td>
</tr>
<tr>
<td>-E behavior</td>
<td>Control what happens if an error is encountered while reading data from the XML file. Specify one of the following supported behavior values:</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--onerror behavior</td>
<td></td>
</tr>
<tr>
<td>● continue</td>
<td>unload ignores the error and continues to load XML.</td>
</tr>
<tr>
<td>● prompt</td>
<td>unload prompts you to continue.</td>
</tr>
<tr>
<td>● quit</td>
<td>unload stops loading the XML and terminates with an error. This behavior is the default behavior if none is specified.</td>
</tr>
<tr>
<td>● exit</td>
<td>unload exits.</td>
</tr>
<tr>
<td>-f directory</td>
<td>Set the directory that contains files with additional data to load. See “UltraLite Database Unload utility (ulunload)” on page 216.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--filedir directory</td>
<td></td>
</tr>
<tr>
<td>-g ID</td>
<td>Set the initial database ID to the INTEGER value you assign. This initial value is used with a partition size for new rows that have global autoincrement columns. When deploying an application, you must assign a different range of identification numbers to each database for synchronization with the MobiLink server. See “UltraLite global_database_id option” on page 184.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--databaseid ID</td>
<td></td>
</tr>
<tr>
<td>-i</td>
<td>Include inserted rows in the next upload synchronization. By default, rows inserted by this utility are not uploaded during synchronization.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--insertforsync</td>
<td></td>
</tr>
<tr>
<td>--identity-file = file</td>
<td>Specify the file containing the client TLS identity. See “identity” [MobiLink - Client Administration].</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>--identity-password = password</strong></td>
<td>Specify the password for the client TLS identity. See “identity_password” [MobiLink - Client Administration].</td>
</tr>
<tr>
<td>-l filename OR --log = filename</td>
<td>Log operations to the specified file.</td>
</tr>
<tr>
<td>-n OR --schemaonly</td>
<td>Load schema metadata only, ignoring any data in the XML input file.</td>
</tr>
<tr>
<td>-o value OR --obfuscate = value</td>
<td>On or off, 1 or 0, and so on. Controls whether data in the database is obfuscated. Obfuscation is a form of simple encryption. See “Database security” on page 26 and “UltraLite obfuscate creation parameter” on page 143.</td>
</tr>
<tr>
<td>-p page-size OR --page_size = page-size</td>
<td>Defines the database page size. See “UltraLite page_size creation parameter” on page 144.</td>
</tr>
<tr>
<td>-q OR --quiet</td>
<td>Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.</td>
</tr>
<tr>
<td>-s file OR --sql = file</td>
<td>Log the SQL statements used to load the database into the specified file.</td>
</tr>
<tr>
<td>-t file OR --rootcert = file</td>
<td>Specify the file containing the trusted root certificate. This certificate is required for server authentication.</td>
</tr>
</tbody>
</table>
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--utf8_encoding=value</td>
<td>On or off, 1 or 0, and so on. Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode. See “UltraLite character sets” on page 24 and “UltraLite utf8_encoding creation parameter” on page 155.</td>
</tr>
<tr>
<td>-v</td>
<td>Print verbose messages.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--verbose</td>
<td></td>
</tr>
<tr>
<td>-y</td>
<td>Overwrite the database file without confirmation. This only applies when you use ulload to create a new database.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--overwrite</td>
<td></td>
</tr>
<tr>
<td>-?</td>
<td>Display the utility usage and exit.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
</tbody>
</table>

### Remarks

The ulload utility takes an input XML file generated by ulunload, ulunloadold (provided with SQL Anywhere 10), or ulxml (in UltraLite versions 8 and 9). When used along with ulunload this utility provides you with the ability to rebuild a database. An alternative method to rebuild a database is using ulunload to generate SQL statements and then use DBISQL to read them into a new database.

The XML file can contain metadata for the schema and/or metadata for the database data. -d ignores the schema metadata, only adding data to the .udb file. -n ignores the data and the metadata, only adding the schema to the .udb file.

Setting an option or specifying a certificate on the command line overrides any settings in the xml-file that is processed by ulload.

The ulload utility restores any synchronization profiles to the database when reading the XML.

This utility returns error codes. Any value other than 0 means that the operation failed.

### See also

- “UltraLite connection parameters” on page 156
- “UltraLite Database Unload utility (ulunload)” on page 216
- “Supported exit codes” on page 187
- “UltraLite global_database_id option” on page 184
Example

Create a new UltraLite database file, sample.udb, and load it with data in sample.xml:

```
ulload -c DBF=sample.udb sample.xml
```

Load the data from sample.xml into the existing database sample.udb, and if an error occurs, prompt for action:

```
ulload -d -c DBF=sample.udb --onerror=prompt sample.xml
```

Create the schema and data stored in test_data.xml in the sample.udb database. Since the -a switch is specified, sample.udb must exist prior to running this command. Moreover, any schema or data that conflicts with what is already in sample.udb will mean the ULLOAD command will fail.

```
ulload -c DBF=sample.udb -a test_data.xml
```

---

**UltraLite Synchronization utility (ulsync)**

Synchronizes an UltraLite database with a MobiLink server. This tool can be used for testing synchronization during application development.

**Syntax**

```
ulsync -c [ options ] [ synchronization parameters]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used. See “Configuration files” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>-c &quot;connection-string&quot;</td>
<td>Required. Connect to the database as identified in the DBF or file_name parameter of your connection-string. If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.</td>
</tr>
<tr>
<td>--connect=&quot;connection-string&quot;</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-p profile-name</td>
<td>Synchronize using the named sync profile, equivalent to: ( \text{SYNCHRONIZE profileName MERGE syncOptions} ) where sync options are taken from the trailing ulsync options. For example: ulsync -p profileName &quot;MobiLinkUid=ml;ScriptVersion=Version 001...syncOptions&quot; See “Synchronization profile options” on page 213.</td>
</tr>
<tr>
<td>-q</td>
<td>Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.</td>
</tr>
<tr>
<td>-r</td>
<td>Display last synchronization results and exit.</td>
</tr>
<tr>
<td>-v</td>
<td>Display synchronization progress messages. This also determines whether progress is displayed for any synchronization, whether using the C++ API or the SQL SYNCHRONIZE PROFILE statement. See “CREATE SYNCHRONIZATION PROFILE statement [UltraLite]” on page 410.</td>
</tr>
<tr>
<td>--log filename</td>
<td>Log operations to the specified file.</td>
</tr>
<tr>
<td>-?</td>
<td>Displays utility usage information and exits.</td>
</tr>
</tbody>
</table>

Remarks
When a certificate file is specified either with the trusted_certificate or the e2ee_public_key option, the UltraLite runtime only looks for these files in the main resource bundle, which is a part of every iPhone application deployment package. You add items to this bundle by including them in the /Resources folder in your Xcode project. This is not applicable to certificates that are stored in the UltraLite database, and
does not affect Mac OS clients (only iPhone). See “trusted_certificates” [MobiLink - Client Administration].

The following options that were valid for versions 10 and earlier are no longer supported: -a authenticate-parameters, -e sync-parms, -k stream-type, -n (no sync), and -x protocol options. -e keyword=value is now part of the sync parameters string and -k and -x are now part of the Stream= stream{stream-parms} sync parameters string.

ulsync can be considered to be equivalent to one of the following SQL statements, depending on usage:

```
ulsync -p profile "parms"
```

is equivalent to:

```
SYNCHRONIZE PROFILE profile MERGE parms
```

and

```
ulsync "parms"
```

is equivalent to:

```
SYNCHRONIZE USING <parms>
```

For secure synchronization, the UltraLite application must have access to the public certificate. You can reference a certificate by:

- Incorporating the certificate information into the UltraLite database at creation time with the -t file option using ulinit or ulload.

- Referencing an external certificate file at synchronization time with the trusted_certificate=file stream option.

This utility returns error codes. Any value other than 0 means that the operation failed.

See also
- “Synchronization profile options” on page 213
- “End-to-end encryption” [SQL Anywhere Server - Database Administration]
- “trusted_certificates” [MobiLink - Client Administration]
- “UltraLite connection parameters” on page 156
- “UltraLite clients” on page 63
- “Supported exit codes” on page 187
- “MobiLink File Transfer utility (mlfiletransfer)” [MobiLink - Client Administration]

Examples

The following command synchronizes a database file called myuldb.udb for a MobiLink user called remoteA.

```
ulsync -c DBF=myuldb.udb "MobiLinkUid=remoteA;Stream=http;ScriptVersion=2"
```

The following command synchronizes a database file called myuldb.udb over HTTPS with the C:\Users \Public\Documents\SQL Anywhere 16\Samples\Certificates\rsaroot.crt certificate. The
trusted_certificate=file option must be used because the trusted certificate file was not added to the database when the database was created. Additionally, the MobiLink user name is remoteB.

ulsync -c DBF=myuldb.udb "Stream=https{trusted_certificate=C:\Users\Public\Documents\SQL Anywhere 16\Samples\Certificates\rsaroot.crt}; MobiLinkUid=remoteB;ScriptVersion=2;UploadOnly=ON"

The following command displays the last synchronization results for a database file named synced.udb.

ulsync -r -c dbf=synced.udb

The previous synchronization results are listed as follows:

SQL Anywhere UltraLite Database Synchronize Utility Version XX.X
Results of last synchronization:
Succeeded
  Upload OK
  No ignored rows
  Partial download retained
  Authentication value: 1000 (0x3e8)

The following example shows the command line used to synchronize the CustDB database with a user name of 50 over TCP/IP on a port of 2439. It uses verbose progress messages.

ulsync -c "dbf=C:\Users\Public\Documents\SQL Anywhere 16\Samples\UltraLite\custdb.udb"
"MobiLinkUid=50;ScriptVersion=custdb 12.0;Stream=tcpip(port=2439)"

The following command illustrates how to use TLS encryption with E2EE:

ulsync -c "uid=dba;pwd=sql;dbf=myudb.db"
"MobiLinkUid=rem1;MobiLinkPwd=password;ScriptVersion=v1;Stream=tls{host=myServer;port=2439;trusted_certificate=c:\clientcert.pem;e2ee_public_key=c:\e2eepublic.pem}"
<table>
<thead>
<tr>
<th>Synchronization profile option</th>
<th>Valid values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthParms</td>
<td>String (comma separated)</td>
<td>Specifies the list of authentication parameters sent to the MobiLink server. You can use authentication parameters to perform custom authentication in MobiLink scripts. See “Authentication Parameters synchronization parameter” on page 85.</td>
</tr>
<tr>
<td>CheckpointStore</td>
<td>Boolean</td>
<td>Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process. See “Additional Parameters synchronization parameter” on page 83.</td>
</tr>
<tr>
<td>ContinueDownload</td>
<td>Boolean</td>
<td>Restarts a previously failed download. When continuing a download, only the changes that were selected to be downloaded with the failed synchronization are received. By default, UltraLite does not continue downloads. See “Resumption of failed downloads” [MobiLink - Server Administration].</td>
</tr>
<tr>
<td>DisableConcurrency</td>
<td>Boolean</td>
<td>Disallow database access from other threads during synchronization. See “Additional Parameters synchronization parameter” on page 83.</td>
</tr>
<tr>
<td>DownloadOnly</td>
<td>Boolean</td>
<td>Performs a download-only synchronization. See “Download Only synchronization parameter” on page 88.</td>
</tr>
<tr>
<td>KeepPartialDownload</td>
<td>Boolean</td>
<td>Controls whether UltraLite keeps a partial download if a communication error occurs. By default, UltraLite does not roll back partially downloaded changes. See “Keep Partial Download synchronization parameter” on page 89.</td>
</tr>
<tr>
<td>MobiLinkPwd</td>
<td>String</td>
<td>Specifies the existing MobiLink password associated with the user name. See “MobiLinkPwd (mp) extended option” [MobiLink - Client Administration].</td>
</tr>
<tr>
<td>MobiLinkUid</td>
<td>String</td>
<td>Specifies the MobiLink user name. See “-u dbmlsync option (deprecated)” [MobiLink - Client Administration] and “-mn dbmlsync option” [MobiLink - Client Administration].</td>
</tr>
<tr>
<td>NewMobiLinkPwd</td>
<td>String</td>
<td>Supplies a new password for the MobiLink user. Use this option when you want to change an existing password. See “-mn dbmlsync option” [MobiLink - Client Administration].</td>
</tr>
<tr>
<td>Ping</td>
<td>Boolean</td>
<td>Confirms communications with the server only; no synchronization is performed. See “Ping synchronization parameter” on page 93.</td>
</tr>
<tr>
<td>Synchronization profile option</td>
<td>Valid values</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Publications</td>
<td>String (comma separated)</td>
<td>Specifies the publications(s) to synchronize. The publications determine the tables on the remote database that are involved in synchronization. If this parameter is blank (the default) then all tables are synchronized. If the parameter is an asterisk (*) then all publications are synchronized. See “Publishing data in UltraLite” on page 72.</td>
</tr>
<tr>
<td>ScriptVersion</td>
<td>String</td>
<td>Specifies the MobiLink script version. The script version determines which scripts are run by MobiLink on the consolidated database during synchronization. If you do not specify a script version, 'default' is used. See “ScriptVersion (sv) extended option” [MobiLink - Client Administration].</td>
</tr>
<tr>
<td>SendDownloadACK</td>
<td>Boolean</td>
<td>Specifies that a download acknowledgement should be sent from the client to the server. By default, the MobiLink server does not provide a download acknowledgement. See “Send Download Acknowledgement synchronization parameter” on page 96.</td>
</tr>
<tr>
<td>Stream</td>
<td>String (with sub-list)</td>
<td>Specifies the MobiLink network synchronization protocol. See “Stream Type synchronization parameter” on page 98.</td>
</tr>
<tr>
<td>TableOrder</td>
<td>String (comma separated)</td>
<td>Specifies the order of tables in the upload. By default, UltraLite selects an order based on foreign key relationships. See “Additional Parameters synchronization parameter” on page 83.</td>
</tr>
<tr>
<td>UploadOnly</td>
<td>String</td>
<td>Specifies that synchronization will only include an upload, and no download will occur. See “Upload Only synchronization parameter” on page 101.</td>
</tr>
</tbody>
</table>

The Boolean values can be specified as Yes/No, 1/0, True/False, On/Off. In all the Boolean cases, the default is No. For all other values, the default is simply unspecified.

See also
- “ALTER SYNCHRONIZATION PROFILE statement [UltraLite]” on page 399
- “DROP SYNCHRONIZATION PROFILE statement [UltraLite]” on page 420
- “SYNCHRONIZE statement [UltraLite]” on page 434
- “UltraLite creation parameters” on page 131
UltraLite Database Unload utility (ulunload)

Unloads any of the following, depending on the options used:

- An entire UltraLite database to XML or SQL.
- All or part of UltraLite data only to XML or SQL.

Syntax

```
ulunload -c "connection-string" [ options ] output-file
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.</td>
</tr>
<tr>
<td>-b max-size</td>
<td>Set the maximum size of column data to be stored in the XML file. The default is 10 KB. To store all data in the XML file (no maximum size), use -b -1.</td>
</tr>
<tr>
<td>--maxblob=max-size</td>
<td></td>
</tr>
<tr>
<td>-c &quot;connection-string&quot;</td>
<td>Required. Connect to the database as identified in the DBF or file_name parameter of your connection-string. If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.</td>
</tr>
<tr>
<td>-d</td>
<td>Only unload the data from the database to the output file. Do not unload any schema information.</td>
</tr>
<tr>
<td>--dataonly</td>
<td></td>
</tr>
<tr>
<td>-e table,...</td>
<td>Exclude the named table when unloading the database. You can name multiple tables in a comma-separated list. For example: -e mydbtable1,mydbtable5</td>
</tr>
<tr>
<td>--exclude=table,...</td>
<td></td>
</tr>
<tr>
<td>-f directory</td>
<td>Set the directory to store data larger than the maximum size specified by -b. The default is the same directory as the output file.</td>
</tr>
<tr>
<td>--filedir=directory</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-l filename</td>
<td>Log operations to the specified file.</td>
</tr>
</tbody>
</table>
| OR
--log=filename |             |
| -n            | Unload schema only, ignoring any data in the database. |
| OR
--schemaonly  |             |
| -q            | Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however. |
| OR
--quiet       |             |
| -s            | Unload as SQL Anywhere-compatible SQL statements. SQL file output can be read by UltraLite or SQL Anywhere using DBISQL. |
| OR
--sql         |             |
| -t table,...  | Unload data in the named table only. You can name multiple tables in a comma separated list. For example: -t mydbtable2,mydbtable6 |
| OR
--include=table,... |             |
| -v            | Print verbose messages. |
| OR
--verbose     |             |
| -x owner      | Output tables so they are owned by a specific user ID. You can use this option with the -s option. |
| OR
--owner=owner |             |
| -y            | Overwrite output-file without confirmation. |
| OR
--overwrite   |             |
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-? OR --help</td>
<td>Displays utility usage information and exits.</td>
</tr>
<tr>
<td>output-file</td>
<td>Required. Set the name of the file that the database is unloaded into. If you use the -s option, database is unloaded as SQL statements. Otherwise, the database is unloaded as XML.</td>
</tr>
</tbody>
</table>

**Remarks**

By default, ulunload outputs XML that describes the schema and data in the database. You can use the output for archival purposes, or to keep the UltraLite database portable across all releases.

Saving a database with a synchronization profile results in XML that is incompatible with earlier versions of the UltraLite utilities. A workaround is to edit the XML and remove the text section marked with

<syncprofiles>...</syncprofiles>

Unloading a database does not preserve:

- Synchronization state, stored synchronization counts, and row deletions. Ensure you synchronize the database before unloading it.

- UltraLite user entries.

To confirm what database options or properties have been preserved, run ulinfo after you have reloaded your database with the ulload utility.

If column data exceeds the maximum size you specified with -b, the overflow is saved to a *.bin file in either:

- the same directory as the XML file
- the directory specified by -f.

The file follows this naming convention:

`tablename-columnname-rownumber.bin`

The -x option allows you to assign ownership to UltraLite tables. You only need to assign an owner to a table if you intend to use the resulting SQL statements for creating or modifying a SQL Anywhere database. When read by UltraLite, the owner names are silently ignored.

This utility returns error codes. Any value other than 0 means that the operation failed.

If you are using this utility to unload a database on the Windows Mobile device directly, UltraLite cannot back up the database before the unload or action occurs. You must perform this action manually before running these wizards.
See also

- “Configuration files” [SQL Anywhere Server - Database Administration]
- “UltraLite connection parameters” on page 156
- “Supported exit codes” on page 187
- “UltraLite Load XML to Database utility (ulload)” on page 206
- “UltraLite Information utility (ulinfo)” on page 197

Example

Unload the sample.udb database into the sample.xml file.

    ulunload -c DBF=sample.udb sample.xml

Unload the data from the sample.udb database into a SQL file called sample1.sql. Overwrite the SQL file if it exists.

    ulunload -c DBF=sample.udb -d -y -s sample.sql

UltraLite Validate Database utility (ulvalid)

Performs a full (normal) validation of an UltraLite database.

Syntax

    ulvalid -c "connection-string" [ options ]

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.</td>
</tr>
<tr>
<td>-c &quot;connection-string&quot;</td>
<td>Required. Connect to the database as identified in connection-string. If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.</td>
</tr>
<tr>
<td>-e</td>
<td>Express validation. Only perform table validation. This option provides a faster validation than normal validation.</td>
</tr>
<tr>
<td>-q</td>
<td>Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.</td>
</tr>
<tr>
<td>--connect=&quot;connection-string&quot;</td>
<td>Required. Connect to the database as identified in connection-string. If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.</td>
</tr>
<tr>
<td>--express</td>
<td>Express validation. Only perform table validation. This option provides a faster validation than normal validation.</td>
</tr>
<tr>
<td>--quiet</td>
<td>Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>-v</td>
<td>Print verbose messages.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--verbose</td>
<td></td>
</tr>
<tr>
<td>--log=filename</td>
<td>Log operations to the specified file.</td>
</tr>
<tr>
<td>-?</td>
<td>Displays utility usage information and exits.</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>--help</td>
<td></td>
</tr>
</tbody>
</table>

**Remarks**

Validating a database verifies the accuracy of the table metadata and ensures the file has not been corrupted.

The validation includes:

- **Database pages**  Validate all database pages, using checksums when enabled. Certain critical pages always have checksums and even pages without checksums undergo a basic validity check.
- **Tables** Validate table(s) by checking that the table row count matches the count in each index.
- **Indexes** Validate indexes by checking that entries refer to valid rows. `ulvalid -e` performs an express check, which includes only table validation.

**See also**

- “UltraLite checksum_level creation parameter” on page 134
- “Configuration files” [*SQL Anywhere Server - Database Administration*]
- “Validating an UltraLite database” on page 41

**Example**

An example of an express validation of a database named `sample.udb` run in quiet mode.

```
ulvalid -c DBF=sample.udb -e -q
```

**UltraLite system tables**

The schema of an UltraLite database is stored in a proprietary format. Earlier versions of UltraLite databases were stored in several system tables. These system tables can still be queried for backward compatibility (they are in essence system views), but they only contain information about user schema (like tables, columns, indexes) not system schema. For example, you cannot query systable to find the properties of systable itself. You can only query systable to find the properties of user-created tables.
Each UltraLite programming API supports objects and methods that can be used to query the database about its schema. It is recommended that you use these objects and APIs to explore schema rather than querying the system views.

All queries performed on these system views are equivalent to full table scans. Index scans are not supported on these system views.

**sysarticle system table**

Each row in the sysarticle system table describes a table that belongs to a publication.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>UNSIGNED INT</td>
<td>An identifier for the publication that this article belongs to.</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>The identifier of the table that belongs to the publication.</td>
</tr>
<tr>
<td>where_expr</td>
<td>VARCHAR(256)</td>
<td>An optional predicate to filter rows.</td>
</tr>
</tbody>
</table>

**Constraints**

PRIMARY KEY (publication_id, table_id)

FOREIGN KEY (publication_id) REFERENCES syspublication (publication_id)

FOREIGN KEY (table_id) REFERENCES systable (object_id)

**See also**

- “syspublication system table” on page 224

**syscolumn system table**

Each row in the syscolumn system table describes one column.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_name</td>
<td>VARCHAR(128)</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>default</td>
<td>VARCHAR(128)</td>
<td>The default value for this column. For example, autoincrement.</td>
</tr>
<tr>
<td>domain</td>
<td>UNSIGNED INT</td>
<td>The column domain, which is an enumerated value indicating the domain of the column.</td>
</tr>
<tr>
<td>domain_info</td>
<td>UNSIGNED INT</td>
<td>Used with a variable sized domain.</td>
</tr>
<tr>
<td>nulls</td>
<td>VARCHAR(1)</td>
<td>Determines if the column allows nulls default.</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>object_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifier for that column.</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>The identifier of the table to which the column belongs.</td>
</tr>
</tbody>
</table>

**Constraints**

- PRIMARY KEY( table_id, object_id )
- FOREIGN KEY (table_id) REFERENCES systable (object_id)

**sysindex system table**

Each row in the sysindex system table describes one index in the database.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_on_commit</td>
<td>BIT</td>
<td>Indicates when referential integrity is checked to ensure every foreign key.</td>
</tr>
<tr>
<td>index_name</td>
<td>VARCHAR(128)</td>
<td>The name of the index.</td>
</tr>
<tr>
<td>ixcol_count</td>
<td>UNSIGNED INT</td>
<td>The number of columns in the index.</td>
</tr>
<tr>
<td>nullable</td>
<td>BIT</td>
<td>Only required if type is foreign. Indicates if nulls are allowed.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifier for an index.</td>
</tr>
<tr>
<td>primary_index_id</td>
<td>UNSIGNED INT</td>
<td>Only required if type is foreign. Lists the identifier of the primary index.</td>
</tr>
<tr>
<td>primary_table_id</td>
<td>UNSIGNED INT</td>
<td>Only required if type is foreign. Lists the identifier of the primary table.</td>
</tr>
<tr>
<td>root_handle</td>
<td>UNSIGNED INT</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifier for the table to which the index applies.</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>VARCHAR(10)</td>
<td>The type of index. Can be one of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• primary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• foreign</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• unique</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• index</td>
</tr>
<tr>
<td>hash_size</td>
<td>UNSIGNED SHORTINT</td>
<td>Stores the hash size used for index hashing.</td>
</tr>
</tbody>
</table>

### Constraints

- PRIMARY KEY (table_id, object_id)
- FOREIGN KEY( table_id ) REFERENCES systable( object_id )

### See also
- “sysixcol system table” on page 223

#### sysixcol system table

Each row in the sysixcol system table describes one column of an index listed in sysindex.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifier for the column being indexed.</td>
</tr>
<tr>
<td>index_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifier for the index that this index-column belongs to.</td>
</tr>
<tr>
<td>order</td>
<td>VARCHAR(1)</td>
<td>Indicates whether the column in the index is kept in ascending (A) or descending (D) order.</td>
</tr>
<tr>
<td>sequence</td>
<td>UNSIGNED INT</td>
<td>The order of the column in the index.</td>
</tr>
<tr>
<td>table_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifier for the table to which the index applies.</td>
</tr>
</tbody>
</table>

### Constraints

- PRIMARY KEY( table_id, index_id, sequence )
- FOREIGN KEY( table_id, index_id ) REFERENCES sysindex( table_id, object_id )
- FOREIGN KEY( table_id, column_id ) REFERENCES syscolumn( table_id, object_id )
See also

- “sysindex system table” on page 222

**syspublication system table**

Each row in the syspublication system table describes a publication.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>download_timestamp</td>
<td>TIMESTAMP</td>
<td>The time of the last download.</td>
</tr>
<tr>
<td>last_sync</td>
<td>UNSIGNED BIGINT</td>
<td>Used to keep track of upload progress.</td>
</tr>
<tr>
<td>publication_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifier for the publication.</td>
</tr>
<tr>
<td>publication_name</td>
<td>VARCHAR(128)</td>
<td>The name of the publication.</td>
</tr>
</tbody>
</table>

**Constraints**

PRIMARY KEY (publication_id)

See also

- “sysarticle system table” on page 221

**syssyncresult system table**

Any row in the syssyncresult system table contains information about the most recent synchronization.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sql_code</td>
<td>INTEGER</td>
<td>The SQL code from the last synchronization.</td>
</tr>
<tr>
<td>error_string</td>
<td>CHAR(200)</td>
<td>The error message from the last synchronization.</td>
</tr>
<tr>
<td>stream_error_code</td>
<td>SMALLINT</td>
<td>The specific stream error. See the ss_error_code enumeration for possible values.</td>
</tr>
<tr>
<td>system_error_code</td>
<td>INTEGER</td>
<td>A system-specific error code. For more information about error codes, see your platform documentation.</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>stream_error_string</td>
<td>CHAR(80)</td>
<td>A string with additional information, if available, for the stream_error_code value.</td>
</tr>
<tr>
<td>upload_ok</td>
<td>BIT</td>
<td>True if the upload was successful; false otherwise.</td>
</tr>
<tr>
<td>ignored_rows</td>
<td>BIT</td>
<td>True if uploaded rows were ignored; false otherwise.</td>
</tr>
<tr>
<td>auth_status</td>
<td>UNSIGNED SMALLINT</td>
<td>The synchronization authentication status.</td>
</tr>
<tr>
<td>auth_value</td>
<td>INTEGER</td>
<td>The value used by the MobiLink server to determine the auth_status result.</td>
</tr>
<tr>
<td>auth_info</td>
<td>CHAR(1024)</td>
<td>The authentication message returned from the MobiLink user authentication script.</td>
</tr>
<tr>
<td>partial_download_retained</td>
<td>BIT</td>
<td>The value that tells you whether a partial download was retained.</td>
</tr>
<tr>
<td>timestamp</td>
<td>TIMESTAMP</td>
<td>The time and date of the last synchronization.</td>
</tr>
<tr>
<td>sent_bytes</td>
<td>UNSIGNED INT</td>
<td>The number of bytes currently sent for the upload.</td>
</tr>
<tr>
<td>sent_inserts</td>
<td>UNSIGNED INT</td>
<td>The number of rows currently inserted for the upload.</td>
</tr>
<tr>
<td>sent_updates</td>
<td>UNSIGNED INT</td>
<td>The number of updated rows currently sent for the upload.</td>
</tr>
<tr>
<td>sent_deletes</td>
<td>UNSIGNED INT</td>
<td>The number of deleted rows currently sent for the upload.</td>
</tr>
<tr>
<td>received_bytes</td>
<td>UNSIGNED INT</td>
<td>The number of bytes currently sent for the download.</td>
</tr>
<tr>
<td>received_inserts</td>
<td>UNSIGNED INT</td>
<td>The number of rows currently inserted for the download.</td>
</tr>
<tr>
<td>received_updates</td>
<td>UNSIGNED INT</td>
<td>The number of updated rows currently sent for the download.</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>received_ignored_updates</td>
<td>UNSIGNED INT</td>
<td>The number of duplicate rows that were received in the download.</td>
</tr>
<tr>
<td>received_deletes</td>
<td>UNSIGNED INT</td>
<td>The number of deleted rows currently sent for the download.</td>
</tr>
<tr>
<td>received_ignored_deletes</td>
<td>UNSIGNED INT</td>
<td>The number of deleted rows that were received in the download of rows that have already been deleted.</td>
</tr>
<tr>
<td>received_truncate_deletes</td>
<td>UNSIGNED INT</td>
<td>The number of rows that were deleted in the download by a truncate operation.</td>
</tr>
</tbody>
</table>

**See also**

- “Keep Partial Download synchronization parameter” on page 89

**systable system table**

Each row in the systable system table describes one table in the database.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_count</td>
<td>UNSIGNED INT</td>
<td>The number of columns in the table.</td>
</tr>
<tr>
<td>index_count</td>
<td>UNSIGNED INT</td>
<td>The number of indexes in the table.</td>
</tr>
<tr>
<td>ixcol_count</td>
<td>UNSIGNED INT</td>
<td>The total number of columns in all indexes in the table.</td>
</tr>
<tr>
<td>table_name</td>
<td>VARCHAR(128)</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>object_id</td>
<td>UNSIGNED INT</td>
<td>A unique identifier for that table.</td>
</tr>
<tr>
<td>sync_type</td>
<td>VARCHAR(32)</td>
<td>Used for MobiLink synchronization. Can be one of either no_sync for no synchronization, all_sync to synchronize every row, or normal_sync for synchronize changed rows only.</td>
</tr>
<tr>
<td>table_type</td>
<td>VARCHAR(32)</td>
<td>user to indicate user-created tables.</td>
</tr>
</tbody>
</table>

**Constraints**

PRIMARY KEY (object_id)
UltraLite Java edition database properties

UltraLite Java edition database property values are defined when the database is first created. They can be changed by re-creating the UltraLite Java edition database or editing their corresponding database option, if available.

UltraLite Java edition supports the following database properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blob_file_base_dir</td>
<td>Returns the base directory the database uses to search for BLOB files.</td>
</tr>
<tr>
<td></td>
<td>See also:</td>
</tr>
<tr>
<td></td>
<td>● “UltraLite Java edition blob_file_base_dir option” on page 229</td>
</tr>
<tr>
<td></td>
<td>● “UltraLite Java Edition Database Load utility (ulj-load)” on page 235</td>
</tr>
<tr>
<td>date_format</td>
<td>Returns the date format the database uses for string conversions.</td>
</tr>
<tr>
<td></td>
<td>See “UltraLite date_format creation parameter” on page 136.</td>
</tr>
<tr>
<td>date_order</td>
<td>Returns the date order the database uses for string conversions.</td>
</tr>
<tr>
<td></td>
<td>See “UltraLite date_order creation parameter” on page 138.</td>
</tr>
<tr>
<td>global_database_id</td>
<td>Returns the value of the global_database_id option used for global autoincrement columns. See “UltraLite global_database_id option” on page 184.</td>
</tr>
<tr>
<td>ml_remote_id</td>
<td>Returns the value of the ml_remote_id option that uniquely identifies the database for MobiLink synchronization. See “UltraLite ml_remote_id option” on page 185.</td>
</tr>
<tr>
<td>database_name</td>
<td>Returns the name (or alias) of the database for the current connection. The name returned matches the DBN connection parameter value. If you did not use the DBN connection parameter, the name returned is the database file without the path and extension.</td>
</tr>
<tr>
<td></td>
<td>See also:</td>
</tr>
<tr>
<td></td>
<td>● “UltraLite DBN connection parameter” on page 167</td>
</tr>
<tr>
<td></td>
<td>● “UltraLite DBF connection parameter” on page 165</td>
</tr>
<tr>
<td>nearest_century</td>
<td>Returns the nearest century the database uses for string conversions.</td>
</tr>
<tr>
<td></td>
<td>See “UltraLite nearest_century creation parameter” on page 142.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>page_size</td>
<td>Returns the page size of the database, in bytes. See “UltraLite page_size creation parameter” on page 144.</td>
</tr>
<tr>
<td>precision</td>
<td>Returns the floating-point precision the database uses for string conversions. See “UltraLite precision creation parameter” on page 146.</td>
</tr>
<tr>
<td>scale</td>
<td>Returns the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision during string conversions by the database. See “UltraLite scale creation parameter” on page 147.</td>
</tr>
<tr>
<td>time_format</td>
<td>Returns the time format the database uses for string conversions. See “UltraLite time_format creation parameter” on page 148.</td>
</tr>
<tr>
<td>timestamp_format</td>
<td>Returns the timestamp format the database uses for string conversions. See “UltraLite timestamp_format creation parameter” on page 150.</td>
</tr>
<tr>
<td>timestamp_increment</td>
<td>Returns the minimum difference between two unique timestamps, in microseconds. See “UltraLite timestamp_increment creation parameter” on page 152.</td>
</tr>
<tr>
<td>timestamp_with_time_zone</td>
<td>Returns the timestamp format for TIMESTAMP WITH TIME ZONE values. See “UltraLite timestamp_with_time_zone format creation parameter” on page 153.</td>
</tr>
</tbody>
</table>

See also
- “Reading database properties” on page 35
- “Accessing database options” on page 35
- “UltraLite creation parameters” on page 131
- “UltraLite Java edition database options” on page 228

**UltraLite Java edition database options**

UltraLite Java edition database option values are defined when the database is first created. They can be altered while connected to the database.

This section describes the UltraLite database options that are available. In addition to these options, the following UltraLite database options are supported by UltraLite Java edition databases:

- global_database_id
- ml_remote_id
See also

- “UltraLite global_database_id option” on page 184
- “UltraLite ml_remote_id option” on page 185
- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “UltraLite creation parameters” on page 131

UltraLite Java edition blob_file_base_dir option

Sets the base directory the database uses to search for BLOB files.

Allowed values
String

Default
The default for BlackBerry smartphones is "file:///SDCard/". The default for other platforms is "".

Remarks
The table column you used to store the file name is stored as the string <base_dir>/
<reference_to_external_file>, where <base_dir> is the base directory where the blobfile is located, and
<reference_to_external_file> is the exact string found in the contents column in the XML file.

The base directory can be set initially using the -f option of the UltraLite Java edition database load utility. Relative filenames are resolved with the blob_file_base_dir option value. If the file name does not begin with the prefix file://, UltraLite then prepends the file name with the option value before attempting to open it.

See also

- “UltraLite Java Edition Database Load utility (uljload)” on page 235
- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_BLOB_FILE_BASE_DIR variable [BlackBerry] [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition date_format option

Sets the date format the database uses for string conversions.

Allowed values
String
Default
"YYYY-MM-DD"

Remarks
For more information, see “UltraLite date_format creation parameter” on page 136.

See also
● “Accessing database options” on page 35
● “Reading database properties” on page 35
● “SET OPTION statement [UltraLite]” on page 431
● “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
● “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
● “Connection.OPTION_DATE_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition date_order option
Sets the date order the database uses for string conversions.

Allowed values
String
Default
"YMD"

Remarks
For more information, see “UltraLite date_order creation parameter” on page 138.

See also
● “Accessing database options” on page 35
● “Reading database properties” on page 35
● “SET OPTION statement [UltraLite]” on page 431
● “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
● “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
● “Connection.OPTION_DATE_ORDER variable [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition nearest_century option
Sets the nearest century the database uses for string conversions.

Allowed values
String
Default
"50"
UltraLite Java edition database options

Remarks
For more information, see “UltraLite nearest_century creation parameter” on page 142.

See also
- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_NEAREST_CENTURY variable [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition precision option
Sets the floating-point precision the database uses for string conversions.

Allowed values
String

Default
"30"

Remarks
For more information, see “UltraLite precision creation parameter” on page 146.

See also
- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_PRECISION variable [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition scale option
Sets the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision during string conversions by the database.

Allowed values
String

Default
"6"
UltraLite database reference

Remarks
For more information, see “UltraLite scale creation parameter” on page 147.

See also

- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_SCALE variable [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition time_format option
Sets the time format the database uses for string conversions.

Allowed values
String

Default
"HH:NN:SS.SSS"

Remarks
For more information, see “UltraLite time_format creation parameter” on page 148.

See also

- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_TIME_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition timestamp_format option
Sets the timestamp format the database uses for string conversions.

Allowed values
String

Default
"YYYY-MM-DD HH:NN:SS.SSS"

Remarks
For more information, see “UltraLite timestamp_format creation parameter” on page 150.
See also

- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_TIMESTAMP_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition timestamp_increment option

Sets the minimum difference between two unique timestamps, in microseconds.

Allowed values

String

Default

"1"

Remarks

For more information, see “UltraLite timestamp_increment creation parameter” on page 152.

See also

- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_TIMESTAMP_INCREMENT variable [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition timestamp_with_time_zone_format option

Sets the timestamp format for TIMESTAMP WITH TIME ZONE values.

Allowed values

String

Default

"YYYY-MM-DD HH:NN:SS.SSS+HH:NN"
Remarks

For more information, see “UltraLite timestamp_with_time_zone_format creation parameter” on page 153.

See also

- “Accessing database options” on page 35
- “Reading database properties” on page 35
- “SET OPTION statement [UltraLite]” on page 431
- “Connection.getDatabaseProperty method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.setOption method [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming]

UltraLite Java edition utilities

UltraLite utilities are supplied to perform maintenance and administration tasks on UltraLite Java edition databases.

UltraLite Java Edition Database Information utility (uljinfo)

Displays information about an existing UltraLite Java edition database.

Syntax

uljinfo -c filename -p password [ options ]

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c filename</td>
<td>Required. Specifies the filename of the UltraLite Java edition database to examine.</td>
</tr>
<tr>
<td>-ek key</td>
<td>Specifies the encryption key needed to access the encrypted UltraLite Java edition database.</td>
</tr>
<tr>
<td>-p password</td>
<td>Specifies the password to connect to the UltraLite Java edition database. The default is sql.</td>
</tr>
<tr>
<td>-v</td>
<td>Displays verbose messages.</td>
</tr>
<tr>
<td>-?</td>
<td>Displays command line usage information.</td>
</tr>
</tbody>
</table>

Example

The following is an example of the output from the uljinfo program:

C:\ULj\bin>uljinfo.cmd -c ..\Samples\Demo1.ulj -p sql
SQL Anywhere UltraLite J Database Information Utility
Database name: ..\Samples\Demo1.ulj
Disk file: '..\Samples\Demo1.ulj'
Database ID: 0
Page size: 1024
0 rows for next upload
Date format: YYYY-MM-DD
Date order: YMD
Nearest century: 50
Numeric precision: 30
Numeric scale: 6
Time format: HH:NN:SS.SSS
Timestamp format: YYYY-MM-DD HH:NN:SS.SSS
Timestamp increment: 1
Number of tables: 1
Number of columns: 2
Number of publications: 0
Number of tables that will always be uploaded: 0
Number of tables that are never synchronized: 0
Number of primary keys: 1
Number of foreign keys: 0
Number of indexes: 0
Last download occurred on Thu Jul 05 11:31:05 EDT 2007
Upload OK: true

UltraLite Java Edition Database Load utility (uljload)

Provides the capability to load an UltraLite Java edition database from an XML source file. The XML file is often produced by the uljunload utility and is customizable.

Note
Database configuration parameters cannot be set on the command line. The uljload utility uses the default configuration when a new database is created. To create a database with different configuration parameters, such as a different page size, create the database in Java first and then use uljload -a to load the XML content into the database you created.

Syntax
uljload -c filename -p password [ options ] inputfile

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>Adds information from the XML file to an existing database. If this option is not specified, a new database is created.</td>
</tr>
<tr>
<td>-c filename</td>
<td>Required. Specifies the name of the database file.</td>
</tr>
<tr>
<td>-d</td>
<td>Loads data only; ignores schema information.</td>
</tr>
<tr>
<td>-ek key</td>
<td>Specifies the encryption key needed to access the encrypted UltraLite Java edition database.</td>
</tr>
<tr>
<td>-f directory</td>
<td>Specifies the directory to retrieve data for columns larger than max blob size specified by the -b option of a uljunload operation.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-i</td>
<td>Inserts rows for upload synchronization.</td>
</tr>
<tr>
<td>-n</td>
<td>Loads schema information only; ignores row data.</td>
</tr>
<tr>
<td>-p password</td>
<td>Optional. Specifies the password. The default password is sql.</td>
</tr>
<tr>
<td>-q</td>
<td>Runs the utility in quiet mode—does not display messages.</td>
</tr>
<tr>
<td>-v</td>
<td>Displays verbose messages.</td>
</tr>
<tr>
<td>-y</td>
<td>Overwrites the output file if it exists (and the -a option is not specified).</td>
</tr>
<tr>
<td>-z pagesize</td>
<td>Specifies the page size, in bytes, to create the database with.</td>
</tr>
<tr>
<td>-?</td>
<td>Displays command line usage information.</td>
</tr>
<tr>
<td>inputfile</td>
<td>Specifies the input file that contains XML statements.</td>
</tr>
</tbody>
</table>

**Example for loading blobfile types**

This example assumes that you have applied the following SQL statement to your UltraLite Java edition database and unloaded the file using the UltraLite Java Edition Database Unload utility.

```sql
CREATE TABLE blobfile_example
  file_name CHAR(size) DEFAULT AUTOFILENAME( prefix, extension ),
  file_contents LONG BINARY STORE AS FILE( file_name ) CASCADE DELETE
```

The ulload utility, while encountering a file exported by the uljunload utility, treats the file_name column as a CHAR column and the file_contents column as a LONG BINARY column.

The uljload utility, while encountering the same XML file, reconstructs the blobfile type. The file_name column stores the string `<base_dir>/<reference_to_external_file>`, where `<base_dir>` is the exact string in the `-f` option, and `<reference_to_external_file>` is the exact string found in the file_contents column in the XML file. The newly constructed database then contains valid references to the external files.

The load fails with an error message prompting you to increase the size of the file_name column if the generated file name is too long.

**See also**

- “UltraLite Java Edition Database Unload utility (uljunload)” on page 237

**Example**

The `-?` option displays command line information.

```shell
uljload -?
```

When the uljload utility command line includes the `-?` option, the following usage information appears:

```
SQL Anywhere UltraLiteJ Database Load Utility
Usage: uljload [options] <XML file>
```
Create and load data into a new UltraLiteJ database from <XML file>.

Options:
- `-a` Add to existing database.
- `-c <file>` Database file.
- `-d` Data only — ignore schema.
- `-ek <key>` Encryption key.
- `-f <directory>` Directory for loading .File paths.
- `-i` Insert rows for upload synchronization.
- `-n` Schema only — ignore data.
- `-p` Password to connect to database.
- `-q` Quiet: do not print messages.
- `-v` Verbose messages.
- `-y` Overwrite file if it already exists.
- `-z <page size>` Page size to create the database with.

## UltraLite Java Edition Database Unload utility (uljunload)

Provides the capability to unload an UltraLite Java edition database—either the data, the schema, or both—to an XML file.

### Syntax

`uljunload -c filename -p password [options] outputfile`

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-b</code> <code>max-blob-size</code></td>
<td>Specifies the maximum size (in bytes) of blob/char data output to XML.</td>
</tr>
<tr>
<td><code>-c filename</code></td>
<td>Required. Specifies the name of database file to unload.</td>
</tr>
<tr>
<td><code>-d</code></td>
<td>Unloads data only; does not output schema information.</td>
</tr>
<tr>
<td><code>-e table, ...</code></td>
<td>Excludes data for tables named in list.</td>
</tr>
<tr>
<td><code>-ek key</code></td>
<td>Specifies the encryption key needed to access the encrypted UltraLite Java edition database.</td>
</tr>
<tr>
<td><code>-f directory</code></td>
<td>Specifies the directory to store data for columns larger than max blob size specified with <code>-b</code> option.</td>
</tr>
<tr>
<td><code>-n</code></td>
<td>Unloads schema information only; does not output data.</td>
</tr>
<tr>
<td><code>-p password</code></td>
<td>Specifies the password to connect to database. The default is <code>sql</code>.</td>
</tr>
<tr>
<td><code>-q</code></td>
<td>Runs in quiet mode; does not display messages.</td>
</tr>
<tr>
<td><code>-t table, ...</code></td>
<td>Outputs data for tables named in list only.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Displays verbose messages.</td>
</tr>
<tr>
<td>Options</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>-y</td>
<td>Overwrites output file if it already exists.</td>
</tr>
<tr>
<td>-?</td>
<td>Displays option usage and help information.</td>
</tr>
<tr>
<td>outputfile</td>
<td>Outputs file name (this file contains XML statements that describe the database contents).</td>
</tr>
</tbody>
</table>

**Example for unloading blobfile types**

Apply the following sample SQL statement to your database to unload a blobfile type:

```sql
CREATE TABLE blobfile_example
file_name CHAR(size) DEFAULT AUTOFILENAME( prefix, extension ),
file_contents LONG BINARY STORE AS FILE( file_name ) CASCADE DELETE
```

When applying this example and using this utility, uljunload exports the file_name column as a regular CHAR but with an additional attribute, default_autofilename, which stores the prefix and extension strings in the form 'prefix', 'extension'. The file_contents column is exported as a LONG BINARY but with an additional attribute, filename_col, which stores the name of the referenced file_name column. In the row containing the blobfile columns, the content of the file_name column is unchanged. The file_contents column behaves like an externally stored blob column and has the following form:

```
file_contents.File="tablename-columnname-rownumber.bin"
```

The contents of the file_contents column are saved as .bin files to the location specified by the -f option.

**See also**

- “UltraLite Java Edition Database Unload utility (uljunload)” on page 237

**Example XML file contents**

```xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ul:schema xmlns:ul="urn:ultralite">
  <collation name="1252LATIN1" case_sensitive="no"/>
  <options>
    <option name="dateformat" value="YYYY-MM-DD"/>
    <option name="dateorder" value="YMD"/>
    <option name="nearestcentury" value="50"/>
    <option name="precision" value="30"/>
    <option name="scale" value="6"/>
    <option name="timeformat" value="HH:NN:SS.SSS"/>
    <option name="timestampformat" value="YYYY-MM-DD HH:NN:SS.SSS"/>
    <option name="timestampincrement" value="1"/>
  </options>
  <tables>
    <table name="ULCustomer" sync="changes">
      <columns>
        <column name="cust_id" type="integer" null="no"/>
        <column name="cust_name" type="char(30)" null="yes"/>
      </columns>
      <primarykey>
        <primarycolumn name="cust_id" direction="asc"/>
      </primarykey>
    </indexes/>
  </table>
</schema>
```

238 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
UltraLite Java Edition Database Transfer utility

The uljdbt utility provides the capability to transfer an UltraLite database from a BlackBerry smartphone to an external device, such as a desktop, laptop, or server. In addition, you can delete a database, display database information, or view or email the database transfer log. The utility consists of two applications that must run simultaneously—the UltraLite Java Edition Database Transfer desktop application (uljdbt) and the BlackBerry smartphone client application (ULjDatabaseTransfer.cod).

The desktop application receives UltraLite Java edition databases using a USB or HTTP connection method. When you start the server application, it waits for a BlackBerry smartphone to transfer the database through the specified connection with the client application. The connection is closed either manually through the application interface, when the application times out, or when the transfer is complete.

The BlackBerry Smartphone client application sends UltraLite Java edition databases through a USB cable or a specified TCP port to the desktop application.
Starting the client application

Transfer, delete, view or email an UltraLite Java edition database to an external device using the uljdbtserv utility.

Prerequisites

There are no prerequisites for this task.

Context and remarks

The client application is a signed file located in the UltraLite\UltraLiteJ\BlackBerry4.2 directory of your SQL Anywhere installation.

Task

1. Load ULjDatabaseTransfer.cod from the UltraLite\UltraLiteJ\BlackBerry4.2 directory of your SQL Anywhere installation.

   The client application icon appears in your list of applications.

2. Start the application and press the trackwheel.

3. On the Database Connection screen, complete the following fields:
   - **Database Name**  The name of the database to transfer to the external device. If the name starts with file:// (this is case sensitive), then the client application tries to find the database in the file system; otherwise, it finds the database in the object store.
   - **Database Password**  The database password used to allow data transfer. If you leave Database Password blank, the default password is used.
   - **Encryption Key**  The database encryption key used to encrypt the database. This option is only required when the database is encrypted.

4. Click Next.

Results

The Action screen appears. This is the screen from which you can access all of the client application functionality.

Transferring an UltraLite Java edition database

Transfer a database using the BlackBerry Smartphone client application.

Prerequisites

There are no prerequisites for this task.
Task

1. Run the uljdbt utility, which is located in %SQLANY16%\Bin32\uljdbtserv.cmd, on the BlackBerry smartphone client.

2. On your BlackBerry's Action screen, click the desired connection method (USB or HTTP).

3. For a USB transfer, click USB database transfer. For an HTTP transfer, proceed to the next step.

4. Follow the directions to start the database transfer desktop application.

   **Note**
   To ensure a successful database transfer, make sure that the device or simulator is connected to the BlackBerry Device Manager. For a simulator, make sure that a USB connection is simulated using USB Cable Connected.

   a. Click Next on the client application.
   b. On the desktop application, make sure USB is selected and click Start.

      The BlackBerry smartphone starts transferring the database to the external device. Progress information appears on the desktop application.
   c. Click OK on both the client and desktop applications to close them.

5. For an HTTP transfer, click HTTP database transfer.

   a. On the HTTP Transfer screen, click Next.
   b. Specify the following values:
      - **Host**  The IP address of your desktop.
      - **Port**  The port specified in Connection Properties on the desktop application.
      - **URL Suffix**  The hostname of the server receiving the transfer, including the http:// suffix (this is required).
   c. Click Next.
Receive a database using the UltraLite Java Edition Database Transfer utility.

Prerequisites
There are no prerequisites for this task.

Task
1. Run uljdbtserv.cmd from the Bin32 directory of your SQL Anywhere installation.
2. On the Connect tab, click the desired Connection Method.
3. Under Connection Properties, specify the following values:
   - Port This field only applies to HTTP connections. Type the TCP port number that you want the BlackBerry smartphone to connect to. Usually, this port number matches the port number specified to the UltraLite Java Edition Database Transfer utility running on the BlackBerry smartphone; however, if you are using SSL, then this number can be different.
● **BlackBerry Password**  This field only applies to USB connections. Type the password used to access the connected BlackBerry smartphone when it is locked. Leave this field blank if there is no password.

● **Timeout**  The number of idle minutes before the server application times out and closes the connection.

● **Output**  Specify a file name and location to which to save the transferred database.

4. Click **Start** to open a connection to the BlackBerry smartphone.

   The server application waits until it either times out or establishes a connection. If you have specified an existing file, you are asked whether you want to overwrite it.

   The **Logs** tab provides details on the server status and transfer progress, including error messages.

**Results**

   The database is received.

---

### Deleting an UltraLite Java edition database

Delete a database.

**Prerequisites**

   There are no prerequisites for this task.

**Task**

1. On your BlackBerry's **Action** screen, click **Delete The Database**.

2. In the confirmation window, click **Delete** to delete the database.

3. In the **Database Deleted** window, click **OK** to close the client.

**Results**

   The database is deleted.

---

### Viewing UltraLite Java edition database information

View database information.

**Prerequisites**

   There are no prerequisites for this task.
Task

1. On your BlackBerry's Action screen, click View Database Info. Scroll down to view all the database information.

2. Click Back to return to the Action screen.

Results

You are able to read the database information.

Viewing a log file

View the log file.

Prerequisites

There are no prerequisites for this task.

Task

1. On your BlackBerry's Database Connection screen, display the menu.

2. Click Log. The log screen appears.

3. To email the log file, enter the email address to send the log to and click Send Email. To return to the previous screen, press Enter.

Results

You are able to read the log file.

UltraLite Java edition system tables

This section introduces the system tables that are available to UltraLite Java edition database.

sysarticles system table

Each row in the sysarticles system table describes a table that belongs to a publication.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication _id</td>
<td>UNSIGNED INTEGER</td>
<td>An identifier for the publication that this article belongs to.</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>table_id</td>
<td>UNSIGNED INTEGER</td>
<td>The identifier of the table that belongs to the publication.</td>
</tr>
<tr>
<td>predicate</td>
<td>VARCHAR(256)</td>
<td>The predicate expression specified by the associated WHERE clause of a CREATE PUBLICATION or ALTER PUBLICATION statement.</td>
</tr>
</tbody>
</table>

**Constraints**

PRIMARY KEY (publication_id, table_id)

---

### syscolumn system table

Each row in the syscolumn system table describes a column.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INTEGER</td>
<td>The identifier of the table to which the column belongs.</td>
</tr>
<tr>
<td>column_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique identifier for the column.</td>
</tr>
<tr>
<td>column_name</td>
<td>VARCHAR(128)</td>
<td>The name of the column. See “Domain interface [UltraLite]” [UltraLite - Java Programming].</td>
</tr>
<tr>
<td>column_flags</td>
<td>TINY</td>
<td>A bitwise combination of the following flags describing the attributes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>0x01</strong> Column is in the primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>0x02</strong> Column is nullable.</td>
</tr>
<tr>
<td>column_domain</td>
<td>TINY</td>
<td>The column domain, which is an enumerated value indicating the domain of the column in the low-order 6 bits. The remaining bits are used internally.</td>
</tr>
<tr>
<td>column_length</td>
<td>UNSIGNED SHORT</td>
<td>The column length. For VARCHAR and BINARY type columns, which are defined in the Domain interface, this is the maximum length in bytes. For NUMERIC type columns, the precision is stored in the first byte, and the scale is stored in the second byte.</td>
</tr>
</tbody>
</table>
### Column name

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>column_default_value</td>
<td>VARCHAR(128)</td>
<td>The default value for this column, which is specified by one of the COLUMN_DEFAULT values in the ColumnSchema interface. For example, COLUMN_DEFAULT_AUTOINC denotes an auto-incrementing default value. If a varchar column has DEFAULT AUTOFILENAME specified, then it stores the parameters prefix and extension using the encoding prefix</td>
</tr>
<tr>
<td>filename_colid</td>
<td>UNSIGNED INTEGER</td>
<td>Stores the column ID of the column that contains the file name for a LONG BINARY column declared as STORE AS FILE; otherwise, this column is null.</td>
</tr>
</tbody>
</table>

### Constraints

- PRIMARY KEY (table_id, column_id)

### sysforeignkey system table

Each row in the sysforeignkey system table describes a foreign key that belongs to a table.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INTEGER</td>
<td>The identifier of the table to which the foreign key belongs.</td>
</tr>
<tr>
<td>foreign_table_id</td>
<td>UNSIGNED SHORT</td>
<td>The identifier of the table to which this foreign-key-column refers to.</td>
</tr>
<tr>
<td>foreign_key_id</td>
<td>UNSIGNED INTEGER</td>
<td>The identifier of the foreign key.</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>The name of the foreign key.</td>
</tr>
<tr>
<td>index_name</td>
<td>VARCHAR(128)</td>
<td>The name of the index the foreign key refers to.</td>
</tr>
</tbody>
</table>

### Constraints

- PRIMARY KEY (table_id, foreign_key_id)

### sysfkcol system table

Each row in the sysfkcol system table describes a foreign key column.
### Column Names and Types

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique identifier for the table to which the foreign key applies.</td>
</tr>
<tr>
<td>foreign_key_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique identifier for the foreign key that this column belongs to.</td>
</tr>
<tr>
<td>item_no</td>
<td>UNSIGNED SHORT</td>
<td>The order of the column in the foreign key.</td>
</tr>
<tr>
<td>column_id</td>
<td>UNSIGNED SHORT</td>
<td>A unique identifier of the table column that refers to the foreign column.</td>
</tr>
<tr>
<td>foreign_column_id</td>
<td>UNSIGNED SHORT</td>
<td>A unique identifier of the table column that is being referred to.</td>
</tr>
</tbody>
</table>

### Constraints

PRIMARY KEY (table_id, foreign_key_id, item_no)

### sysindex system table

Each row in the sysindex system table describes an index in the database.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique identifier for the table to which the index applies.</td>
</tr>
<tr>
<td>index_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique identifier for an index.</td>
</tr>
<tr>
<td>index_name</td>
<td>VARCHAR(128)</td>
<td>The name of the index.</td>
</tr>
</tbody>
</table>
| index_flags      | TINY             | A bitwise combination of the following flags denoting the type of index and its persistence:
|                  |                  | ● 0x01  Unique key.                                                        |
|                  |                  | ● 0x02  Unique index.                                                      |
|                  |                  | ● 0x04  Index is persistent.                                              |
|                  |                  | ● 0x08  Primary key.                                                      |
| index_data       | UNSIGNED INTEGER | Internal use only.                                                        |
| hash_size        | UNSIGNED SHORT   | The hash size used for index hashing.                                     |
Constraints

PRIMARY KEY (table_id, index_id)

**sysindexcolumn system table**

Each row in the sysindexcolumn system table describes a column of an index listed in sysindex.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique identifier for the table to which the index applies.</td>
</tr>
<tr>
<td>index_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique identifier for the index that this index-column belongs to.</td>
</tr>
<tr>
<td>order</td>
<td>UNSIGNED INTEGER</td>
<td>The order of the column in the index.</td>
</tr>
<tr>
<td>column_id</td>
<td>UNSIGNED SHORT</td>
<td>A unique identifier for the column being indexed.</td>
</tr>
<tr>
<td>index_column_flag</td>
<td>TINY</td>
<td>An indication of where the column in the index is kept, either in ascending (1) or descending (0) order.</td>
</tr>
</tbody>
</table>

Constraints

PRIMARY KEY (table_id, index_id, order)

**syspublications system table**

Each row in the syspublications system table describes a publication.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique identifier for the publication.</td>
</tr>
<tr>
<td>publication_name</td>
<td>VARCHAR(128)</td>
<td>The name of the publication.</td>
</tr>
<tr>
<td>download_timestamp</td>
<td>TIMESTAMP</td>
<td>The time of the last download.</td>
</tr>
<tr>
<td>last_sync_sent</td>
<td>UNSIGNED INTEGER</td>
<td>An integer that tracks an upload sent to Mobi-Link.</td>
</tr>
<tr>
<td>last_sync_confirmed</td>
<td>UNSIGNED INTEGER</td>
<td>An integer that tracks an upload confirmed as being received by MobiLink.</td>
</tr>
</tbody>
</table>
**Constraints**

PRIMARY KEY (publication_id)

---

**systable system table**

Each row in the systable system table describes a table in the database.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>UNSIGNED INTEGER</td>
<td>A unique identifier for the table.</td>
</tr>
<tr>
<td>table_name</td>
<td>VARCHAR(128)</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>table_flags</td>
<td>UNSIGNED SHORT</td>
<td>A bitwise combination of one of the following flags:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● TableSchema.TABLE_IS_SYSTEM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● TableSchema.TABLE_IS_NO_SYNC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● TableSchema.TABLE_IS_DOWNLOAD_ONLY</td>
</tr>
<tr>
<td>table_data</td>
<td>UNSIGNED INTEGER</td>
<td>Internal use only.</td>
</tr>
<tr>
<td>table_partition_size</td>
<td>UNSIGNED INTEGER</td>
<td>The partition size of the table, if specified or if the table contains a default value; otherwise, null.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value is 0 if the partition size is out of range.</td>
</tr>
</tbody>
</table>

---

**Constraints**

PRIMARY INDEX (table_id)

---

**sysuldata system table**

Each row in the sysuldata system table names value pairs of options and properties.

Synchronization profiles are stored in the sysuldata system table with name set to the profile's name, type set to ulsync, and long_setting set to the UTF-8 encoding of the profile string. Exactly one of the column settings and long_setting has an assigned value depending on usage.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long_setting</td>
<td>LONGBINARY</td>
<td>A BLOB for long values.</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>The name of the 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>setting</td>
<td>VARCHAR(128)</td>
<td>The value of the property.</td>
</tr>
<tr>
<td>type</td>
<td>VARCHAR(128)</td>
<td>One of either opt for options, or prop for properties</td>
</tr>
</tbody>
</table>

Constraints

PRIMARY KEY (name, type)

See also

- “Connection.OPTION_BLOB_FILE_BASE_DIR variable [BlackBerry] [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_DATABASE_ID variable [BlackBerry] [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_DATE_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_DATE_ORDER variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_ML_REMOTE_ID variable [BlackBerry] [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_NEAREST_CENTURY variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_PRECISION variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_SCALE variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_TIME_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_TIMESTAMP_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_TIMESTAMP_INCREMENT variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.OPTION_TIMESTAMP_WITH_TIME_ZONE_FORMAT variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.PROPERTY_DATABASE_NAME variable [UltraLiteJ]” [UltraLite - Java Programming]
- “Connection.PROPERTY_PAGE_SIZE variable [UltraLiteJ]” [UltraLite - Java Programming]
UltraLite SQL reference

This section provides a reference for UltraLite SQL. UltraLite SQL is a unique subset of the SQL supported by SQL Anywhere databases.

UltraLite SQL language elements

Keywords in UltraLite

Each SQL statement contains one or more keywords. SQL keywords are case insensitive, but throughout the documentation, keywords are indicated in uppercase. Some keywords cannot be used as identifiers without surrounding them in double quotes. These are called reserved words.

Note

UltraLite only supports a subset of SQL Anywhere keywords. However, to avoid potential problems in future releases, you should assume that all the reserved words for SQL Anywhere apply to UltraLite as well.

See also

- “Reserved words” [SQL Anywhere Server - SQL Reference]

Identifiers in UltraLite

Identifiers are names of objects in the database, such as user IDs, tables, and columns. Identifiers have a maximum length of 128 bytes.

You must enclose identifiers in double 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 database collation sequence dictates which characters are considered alphabetic or digit characters.
- The identifier contains a reserved word.
- The identifier contains characters other than alphabetic characters and digits.

You can only use a single backslash in an identifier if it is used as an escape character.

See also

- “Reserved words” [SQL Anywhere Server - SQL Reference]
Strings in UltraLite

Strings are used to hold character data in the database. UltraLite supports the same rules for strings as SQL Anywhere. The results of comparisons on strings, and the sort order of strings, depends on the case sensitivity of the database, the character set, and the collation sequence. These properties are set when the database is created.

See also

- “Strings” [SQL Anywhere Server - SQL Reference]
- “UltraLite character sets” on page 24

Comments in UltraLite

Comments are used to attach explanatory text to SQL statements or statement blocks. The UltraLite runtime does not execute comments.

The following comment indicators are available in UltraLite:

- -- (Double hyphen) The database server ignores any remaining characters on the line. This indicator is the SQL/2003 comment indicator.

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

- /* ... */ (Slash-asterisk) Any characters between the two comment markers are ignored. The two comment markers may be on the same or different lines. Comments indicated in this style can be nested. This style of commenting is also called C-style comments.

Note

The percent sign (%) is not supported in UltraLite.

Examples

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

```sql
CREATE TABLE borrowed_book ( 
    loaner_name CHAR(100) PRIMARY KEY, 
    date_borrowed DATE NOT NULL DEFAULT CURRENT DATE, 
    date_returned DATE, 
    book CHAR(20) 
FOREIGN KEY book REFERENCES library_books (isbn), 
) 
--This statement creates a table for a library database to hold information on borrowed books. 
--The default value for date_borrowed indicates that the book is borrowed on the day the entry is made. 
--The date_returned column is NULL until the book is returned.
```

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

```sql
CREATE TABLE borrowed_book ( 
    loaner_name CHAR(100) PRIMARY KEY, 
    date_borrowed DATE NOT NULL DEFAULT CURRENT DATE, 
    date_returned DATE, 
```
Numbers in UltraLite

Numbers are used to hold numerical data in the database. A number can:

- be any sequence of digits
- be appended with decimal parts
- include an optional negative sign (-) or a plus sign (+)
- be followed by an e and then a numerical exponent value

For example, all numbers shown below are supported by UltraLite:

42
-4.038
.001
3.4e10
1e-10

The NULL value in UltraLite

As with SQL Anywhere, 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 unknown (no value) or inapplicable information.

See also

- “NULL special value” [SQL Anywhere Server - SQL Reference]

Special values in UltraLite

You can use special values in expressions, and as column defaults when you create tables.

CURRENT DATE special value

Returns the current year, month, and day.
Data type

DATE

Remarks

The returned date is based on a reading of the system clock when the SQL statement is executed by the UltraLite runtime. If you use CURRENT DATE with any of the following, all values are based on separate clock readings:

- CURRENT DATE multiple times within the same statement
- CURRENT DATE with CURRENT TIME or CURRENT TIMESTAMP within a single statement
- CURRENT DATE with the NOW function or GETDATE function within a single statement

See also

- “Expressions in UltraLite” on page 256
- “GETDATE function [Date and time]” on page 333
- “NOW function [Date and time]” on page 355

CURRENT TIME special value

The current 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 current time is limited by the accuracy of the system clock.

The returned date is based on a reading of the system clock when the SQL statement is executed by the UltraLite runtime. If you use CURRENT TIME with any of the following, all values are based on separate clock readings:

- CURRENT TIME multiple times within the same statement
- CURRENT TIME with CURRENT DATE or CURRENT TIMESTAMP within a single statement
- CURRENT TIME with the NOW function or GETDATE function within a single statement

See also

- “Expressions in UltraLite” on page 256
- “GETDATE function [Date and time]” on page 333
- “NOW function [Date and time]” on page 355
CURRENT TIMESTAMP special value

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 3 decimal places. The accuracy is limited by the accuracy of the system clock.

Columns declared with DEFAULT CURRENT TIMESTAMP do not necessarily contain unique values.

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

CURRENT_TIMESTAMP is equivalent to CURRENT TIMESTAMP.

The returned date is based on a reading of the system clock when the SQL statement is executed by the UltraLite runtime. If you use CURRENT TIMESTAMP with any of the following, all values are based on separate clock readings:

- CURRENT_TIMESTAMP multiple times within the same statement
- CURRENT_TIMESTAMP with CURRENT DATE or CURRENT TIME within a single statement
- CURRENT_TIMESTAMP with the NOW function or GETDATE function within a single statement

See also
- “CURRENT TIME special value” on page 254
- “Expressions in UltraLite” on page 256
- “NOW function [Date and time]” on page 355
- “GETDATE function [Date and time]” on page 333
- “NOW function [Date and time]” on page 355

CURRENT UTC TIMESTAMP special value

Returns a TIMESTAMP WITH TIME ZONE value that reflects the current UTC time containing the year, month, and day.

Data type
DATE

Remarks
The returned date is based on a reading of the system clock when the SQL statement is executed by the UltraLite runtime.
CURRENT DATE multiple times within the same statement

CURRENT DATE with CURRENT TIME or CURRENT TIMESTAMP within a single statement

CURRENT DATE with the NOW function or GETDATE function within a single statement

See also

● “Expressions in UltraLite” on page 256
● “GETDATE function [Date and time]” on page 333
● “CURRENT TIMESTAMP special value” on page 255
● “NOW function [Date and time]” on page 355

SQLCODE special value

Current SQLCODE value at the time the special value was evaluated. Not supported by UltraLite Java edition databases.

Data type

String

Remarks

The SQLCODE value is set after each statement. You can check the SQLCODE to determine if the statement succeeded.

See also

● “Expressions in UltraLite” on page 256
● “Error Messages”

Example

Use a SELECT statement to produce an error code for each attempt to fetch a new row from the result set. For example: SELECT a, b, SQLCODE FROM MyTable.

Dates and times in UltraLite

Many of the date and time functions use dates built from date and time parts. UltraLite and SQL Anywhere support the same date parts.

See also

● “Specifying date parts” on page 295

Expressions in UltraLite

Expressions are formed by combining data, often in the form of column references, with operators or functions.
Syntax

expression:
  case-expression
  constant
  [correlation-name.]column-name
  - expression
  expression operator expression
  ( expression )
  function-name ( expression, ... )
  if-expression
  special value
  input-parameter

Parameters

  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

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

  input-parameter:
  { ? | :name [ : indicator-name ] }

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

- “Constants in expressions” on page 258
- “Special values in UltraLite” on page 253
- “Column names in expressions” on page 258
- “UltraLite SQL functions” on page 294
- “Subqueries in expressions” on page 261
- “Search conditions in UltraLite” on page 263
- “UltraLite SQL data types” on page 275
- “CASE expressions” on page 259
- “Input parameters” on page 262

Constants in expressions

In UltraLite, constants are numbers or string literals.

Syntax

' constant '

Usage

String constants are enclosed in single quotes (').

An apostrophe is represented inside a string by two single quotes in a row ('').

See also

- “Escape sequences” [SQL Anywhere Server - SQL Reference]

Example

To use a possessive phrase, type the string literal as follows:

'John's database'

Column names in expressions

An identifier in an expression.

Syntax

correlation-name.column-name

Remarks

A column name is preceded by an optional correlation name, which typically is the name of a table.

If a column name is a keyword or 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
- “FROM clause [UltraLite]” on page 422

**IF expressions**
Sets a search condition to return a specific subset of data.

**Syntax 1**
```
IF search-condition
THEN expression1
[ ELSE expression2 ]
ENDIF
```

**Remarks**
For compatibility reasons, this expression can end in either ENDIF or END IF.

This expression returns the following:
- If `search-condition` is TRUE, the IF expression returns `expression1`.
- If `search-condition` is FALSE and an ELSE clause is specified, the IF expression returns `expression2`.
- If `search-condition` is FALSE, and there is no `expression2`, the IF expression returns NULL.
- If `search-condition` is UNKNOWN, the IF expression returns NULL.

See also
- “NULL special value” [SQL Anywhere Server - SQL Reference]
- “Search conditions” [SQL Anywhere Server - SQL Reference]

**CASE expressions**
Provides conditional SQL expressions.

**Syntax 1**
```
CASE expression1
WHEN expression2 THEN expression3, ...
[ ELSE expression4 ]
END
```

SELECT ID,
( CASE name
  WHEN 'Tee Shirt' THEN 'Shirt'
  WHEN 'Sweatshirt' THEN 'Shirt'
  WHEN 'Baseball Cap' THEN 'Hat'
  ELSE 'Unknown'
END ) as Type
FROM Product
Syntax 2

    CASE  
    WHEN search-condition  
    THEN expression1, ...  
    [ ELSE expression2 ]  
    END      

Remarks

For compatibility reasons, you can end this expression with either ENDCASE or END CASE.

You can use case expressions anywhere you can use regular expression.

Syntax 1

If the expression following the CASE keyword is equal to the expression following the first WHEN keyword, then the expression following the associated THEN keyword is returned. Otherwise the expression following the ELSE keyword is returned, if specified.

For example, the following code uses a case expression as the second clause in a SELECT statement. It selects a row from the Product table where the name column has a value of Sweatshirt.

Syntax 2

If the search-condition following the first WHEN keyword is TRUE, the expression following the associate THEN keyword is returned. Otherwise the expression following the ELSE clause is returned, if specified.

NULLIF function for abbreviated CASE expressions

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

    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.

Example

The following statement uses a CASE expression as the third clause of a SELECT statement to associate a string with a search condition. If the name column's value is Tee Shirt, this query returns Sale. And if the name column's value is not Tee Shirt and the quantity is greater than fifty, it returns Big Sale. However, for all others, the query then returns Regular price.

    SELECT ID, name,  
    ( CASE  
        WHEN name='Tee Shirt' THEN 'Sale'  
        WHEN quantity >= 50  THEN 'Big Sale'  
        ELSE 'Regular price'  
        END ) as Type  
    FROM Product

Aggregate expressions

Performs an aggregate computation that the UltraLite runtime does not provide.
Syntax

\[
\text{SUM( expression )}
\]

Remarks

An aggregate expression calculates a single value from a range of rows.

An aggregate expression is one in which either an aggregate function is used, or in which one or more of the operands is an aggregate expression.

When a SELECT statement does not have a GROUP BY clause, the expressions in the SELECT list must either contain all aggregate expressions or no aggregate expressions. When a SELECT statement does have a GROUP BY clause, any non-aggregate expression in the SELECT list must appear in the GROUP BY list.

Example

For example, the following query computes the total payroll for employees in the employee table. In this query, \( \text{SUM( salary )} \) is an aggregate expression:

```sql
SELECT SUM( salary )
FROM employee
```

Subqueries in expressions

A SELECT statement that is nested inside another SELECT statement.

Syntax

A subquery is structured like a regular query.

Remarks

In UltraLite, you can only use subquery references in the following situations:

- As a table expression in the FROM clause. This form of table expression (also called derived tables) must have a derived table name and column names in which values in the SELECT list are fetched.

- To supply values for the EXISTS, ANY, ALL, and IN search conditions.

You can write subqueries about names that are specified before (to the left of) the subquery, sometimes known as outer references to the left. However, you cannot have references to items within subqueries (sometimes known as inner references).

See also

- “SELECT statement [UltraLite]” on page 429
- “Use of subqueries” [SQL Anywhere Server - SQL Usage]
- “Search conditions in UltraLite” on page 263

Example

The following subquery is used to list all product IDs for items that are low in stock (that is, less than 20 items).
FROM SalesOrderItems
( SELECT ID
FROM Products
WHERE Quantity < 20 )

Input parameters
Acts as placeholders to allow end-users to supply values to a prepared statement. These user-supplied values are then used to execute the statement.

Syntax
{ ? | :name [ : indicator-name ] }

Remarks
Use the placeholder character of ? or the named form in expressions. You can use input parameters whenever you can use a column name or constant.

The precise mechanism used to supply the values to the statement are dependent upon the API you use to create your UltraLite client.

Using the named form  The named form of an input parameter has special meaning. In general, name is always used to specify multiple locations where an actual value is supplied.

For embedded SQL applications only, the indicator-name supplies the variable into which the null indicator is placed. If you use the named form with the other components, indicator-name is ignored.

Deducing data types  The data type of the input parameter is deduced when the statement is prepared from one of the following patterns:

- CAST ( ? AS type )
  In this case, type is a database type specification such as CHAR(32).

- Exactly one operand of a binary operator is an input parameter. The type is deduced to be the type of the operand.

If the type cannot be deduced, UltraLite generates an error. For example:

- -?: the operand is unary.

- ? + ?: both are input parameters.

See also
- “Host variables” [UltraLite - C and C++ Programming]
- “Prepared statements” [SQL Anywhere Server - Programming]
- UltraLite C/C++: “Data modification using INSERT, UPDATE and DELETE” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Data modification using INSERT, UPDATE, and DELETE” [UltraLite - .NET Programming]
Example

The following embedded SQL statement has two input parameters:

```sql
INSERT INTO MyTable VALUES ( :v1, :v2, :v1)
```

The first instance of v1 supplies its value to both the v2 and v1 locations in the statement.

Search conditions in UltraLite

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

Syntax

```sql
search-condition:
  expression comparison-operator expression
| expression IS [ NOT ] NULL
| expression [ NOT ] BETWEEN expression AND expression
| expression [ NOT ] IN ( expression, ... )
| expression [ NOT ] IN ( subquery )
| expression [ NOT ] { ANY | ALL } ( subquery )
| expression [ NOT ] EXISTS ( subquery )
| expression [ NOT ] LIKE ( pattern )
| NOT search-condition
| search-condition AND search-condition
| search-condition OR search-condition
| ( search-condition IS [ NOT ] { TRUE | FALSE | UNKNOWN } )
```

```sql
comparison-operator :
  =
| >
| <
| >=
| <=
| !=
| !<
| !>
```

Parameters

The different types of search conditions supported by UltraLite include:

- ALL condition
- ANY condition
- BETWEEN condition
- EXISTS condition
- IN condition
- LIKE condition

Remarks

In UltraLite, search conditions can appear in the:
Search conditions can be used to choose a subset of the rows from a table in a FROM clause in a SELECT statement, or in expressions such as an IF or CASE to select specific values. In UltraLite, every condition evaluates as one of three states: TRUE, FALSE, or UNKNOWN. When combined, these states are referred to as **three-valued logic**. The result of a comparison is UNKNOWN if either value being compared is the NULL value. Search conditions are satisfied only if the result of the condition is TRUE.

**Comparison operators**

Any operator that allows two or more expressions to be compared with in a search condition.

**Syntax**

```
expression operator expression
```

**Parameters**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Interpretation</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>
</tbody>
</table>
Operator | Interpretation
---|---
!< | not less than

Remarks

**Comparing dates**  In comparing dates, < means earlier and > means later.

**Comparing LONG VARCHAR or LONG BINARY values**  UltraLite does not support comparisons using LONG VARCHAR or LONG BINARY values.

**Case-sensitivity**  In UltraLite, comparisons are carried out with the same attention to case as the database on which they are operating. By default, UltraLite databases are created as case insensitive.

**NOT operator**  The NOT operator negates an expression.

See also

- “Logical operators” on page 265
- “Search conditions in UltraLite” on page 263

Example

Either of the following two queries will find all Tee shirts and baseball caps that cost $10 or less. However, note the difference in position between the negative logical operator (NOT) and the negative comparison operator (!>).

```sql
SELECT ID, Name, Quantity
FROM Products
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND NOT UnitPrice > 10
```

```sql
SELECT ID, Name, Quantity
FROM Products
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND UnitPrice !> 10
```

**Logical operators**

Does any of the following:

- Compare conditions (AND, OR, and NOT).
- Test the truth or NULL value nature of the expressions (IS).

**Syntax 1**

```
condition1 logical-operator condition2
```

**Syntax 2**

```
NOT condition
```
Syntax 3

expression IS [ NOT ] { truth-value | NULL }

Remarks

Search conditions can be used to choose a subset of the rows from a table in a FROM clause in a SELECT statement, or in expressions such as an IF or CASE to select specific values. In UltraLite, every condition evaluates as one of three states: TRUE, FALSE, or UNKNOWN. When combined, these states are referred to as three-valued logic. The result of a comparison is UNKNOWN if either value being compared is the NULL value. Search conditions are satisfied only if the result of the condition is TRUE.

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

\[ condition1 \text{ AND } condition2 \]

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

\[ condition \]

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

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

See also

- “Three-valued logic” [SQL Anywhere Server - SQL Reference]
- “Comparison operators” on page 264
- “Search conditions in UltraLite” on page 263

Example

The IS NULL condition is satisfied if the column contains a NULL value. If you use the IS NOT NULL operator, the condition is satisfied when the column contains a value that is not NULL. This example shows an IS NULL condition: WHERE paid_date IS NULL.

ALL search condition

Use the ALL condition with comparison operators to compare a single value to the data values produced by the subquery.

Syntax

expression compare [ NOT ] ALL ( subquery )

Parameters

\[ compare:\]

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


Remarks
UltraLite uses the specified comparison operator to compare the test value to each data value in the result set. If all the comparisons yield TRUE results, the ALL test returns TRUE.

See also
● “Subqueries and the ALL test” [SQL Anywhere Server - SQL Usage]
● “Comparison operators” on page 264

Example
Find the order and customer IDs of those orders placed after all products of order #2001 were shipped.

```sql
SELECT ID, CustomerID
FROM SalesOrders
WHERE OrderDate > ALL (SELECT ShipDate
FROM SalesOrderItems
WHERE ID=2001)
```

ANY search condition
Use the ANY condition with comparison operators to compare a single value to the column of data values produced by the subquery.

Syntax 1
```
expression compare [ NOT ] ANY ( subquery )
```

Syntax 2
```
expression = ANY ( subquery )
```

Parameters
\( compare: \)
= | > | < | >= | <= | <> | != | !< | !>

Remarks
UltraLite uses the specified comparison operator to compare the test value to each data value in the column. If any of the comparisons yields a TRUE result, the ANY test returns TRUE.

Syntax 1 is TRUE if \( expression \) is equal to any of the values in the result of the subquery, and FALSE if the expression is not NULL and does not equal any of the values returned by the subquery. The ANY condition is UNKNOWN if \( expression \) is the NULL value, unless the result of the subquery has no rows, in which case the condition is always FALSE.

See also
● “Subqueries and the ANY test” [SQL Anywhere Server - SQL Usage]
● “Comparison operators” on page 264
Example

Find the order and customer IDs of those orders placed after the first product of the order #2005 was shipped.

```
SELECT ID, CustomerID
FROM SalesOrders
WHERE OrderDate > ANY (  
  SELECT ShipDate  
  FROM SalesOrderItems  
  WHERE ID=2005)
```

**BETWEEN search condition**

Specifies an inclusive range, in which the lower value and the upper value and the values they delimit are searched for.

**Syntax**

```
expression [ NOT ] BETWEEN start-expression AND end-expression
```

**Remarks**

The BETWEEN condition can evaluate to TRUE, FALSE, or UNKNOWN. Without the NOT keyword, the condition evaluates as TRUE if `expression` is between `start-expression` and `end-expression`. The NOT keyword reverses the meaning of the condition, but leaves UNKNOWN unchanged.

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

```
[ NOT ] ( expression >= start-expression  
  AND expression <= end-expression )
```

**Example**

List all the products less expensive than $10 or more expensive than $15.

```
SELECT Name, UnitPrice
FROM Products
WHERE UnitPrice NOT BETWEEN 10 AND 15
```

**EXISTS search condition**

Checks whether a subquery produces any rows of query results.

**Syntax**

```
[ NOT ] EXISTS ( subquery )
```

**Remarks**

The EXISTS 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 condition cannot be UNKNOWN.

You can reverse the logic of the EXISTS condition by using the NOT EXISTS form. In this case, the test returns TRUE if the subquery produces no rows, and FALSE otherwise.
Example
List the customers who placed orders after July 13, 2001.

```
SELECT GivenName, Surname
FROM Customers
WHERE EXISTS ( 
  SELECT *
  FROM SalesOrders
  WHERE (OrderDate > '2001-07-13') AND 
    (Customers.ID = SalesOrders.CustomerID)
)
```

**IN search condition**
Checks membership by searching a value from the main query with another value in the subquery.

**Syntax**
```
expression [ NOT ] IN 
{( subquery ) | ( value-expr, ... )}
```

**Parameters**

`value-expr` are expressions that take on a single value, which may be a string, a number, a date, or any other SQL data type.

**Remarks**
An IN condition, without the NOT keyword, 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 `subquery` returns no values; or if `expression` is not NULL, none of the values are NULL, and `expression` does not equal any of the values.

You can reverse the logic of the IN condition by using the NOT IN form.

The following search condition `expression IN ( values )` is identical to the search condition `expression = ANY ( values )`. The search condition `expression NOT IN ( values )` is identical to the search condition `expression <> ALL ( values )`.

**Example**
Select the company name and state for customers who live in the following Canadian provinces: Ontario, Manitoba, and Quebec.

```
SELECT CompanyName , Province
FROM Customers
WHERE State IN( 'ON', 'MB', 'PQ')
```
LIKE search condition

Syntax

The syntax for the LIKE search condition is as follows:

\[ \text{expression} \ [ \text{NOT} \ ] \text{LIKE} \ \text{pattern} \]

Parameters

- **expression**  
  The string to be searched.
- **pattern**  
  The pattern to search for within \textit{expression}.

Remarks

The LIKE search condition attempts to match \textit{expression} with \textit{pattern} and evaluates to TRUE, FALSE, or UNKNOWN. The search condition evaluates to TRUE if \textit{expression} matches \textit{pattern} (assuming NOT was not specified). If either \textit{expression} or \textit{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.

\textit{expression} and \textit{pattern} are interpreted as CHAR strings. \textit{pattern} 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, b1% matches bl and bla.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Any single character in the specified range or set. For example, T[o]m matches Tom or Tim.</td>
</tr>
<tr>
<td>[^]</td>
<td>Any single character not in the specified range or set. For example, M[^c] matches Mb and Md, but not Mc.</td>
</tr>
</tbody>
</table>

Example

The following search condition returns TRUE for any row where column-name starts with the letter a and has the letter b as its second last character:

\[
\text{SELECT} * \text{ FROM table-name WHERE column-name LIKE 'a\%b_'}
\]
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><code>LIKE 'sm[iy]th'</code></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 smith and smyth.</td>
</tr>
<tr>
<td>One of a range of characters</td>
<td><code>LIKE '[a-rt]ough'</code></td>
<td>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 &quot;greater than or equal to a, and less than or equal to z&quot;, 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.</td>
</tr>
<tr>
<td>Ranges and sets combined</td>
<td><code>... LIKE '[a-rt]ough'</code></td>
<td>You can combine ranges and sets within square brackets. In this example, <code>... LIKE '[a-rt]ough'</code> 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.</td>
</tr>
<tr>
<td>One character not in a range</td>
<td><code>... LIKE '[^a-rt]ough'</code></td>
<td>The caret character (^) is used to specify a range of characters that is excluded from a search. In this example, <code>LIKE '[^a-rt]ough'</code> 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.</td>
</tr>
<tr>
<td>Search patterns with trailing blanks</td>
<td>'90 ', '90[ ]' and '90_'</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 [.

- 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:

   LIKE '
([a-zA-Z][oO][uU][gG][hH])

Operators in UltraLite
Operators are used to compute values, which may in turn be used as operands in a higher-level expression.

UltraLite SQL supports the following types of operators:

- Comparison operators evaluate and return a result using one (unary) or two (binary) comparison operands. Comparisons result in the usual three logical values: true, false, and unknown.

- Arithmetic operators evaluate and return a result set for all floating-point, decimal, and integer numbers.

- String operators concatenate two string values together. For example, "my" + "string" returns the string "my string".

- Bitwise operators evaluate and turn specific bits on or off within the internal representation of an integer.

- Logical operators evaluate search conditions. Logical evaluations result in the usual three logical values: true, false, and unknown.

The normal precedence of operations applies.
Arithmetic operators

Arithmetic operators allow you to perform calculations.

- `expression + expression`  Addition. If either expression is NULL, the result is NULL.
- `expression - expression`  Subtraction. If either expression is NULL, the result is NULL.
- `- expression`  Negation. If the expression is NULL, 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. If either expression is NULL, the result is NULL.

See also

- “Arithmetic operations” [SQL Anywhere Server - SQL Usage]

String operators

String operators allow you to concatenate strings—except for LONGVARCHAR and LONGBINARY data types.

- `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:

```
SELECT 123 + 456
```

However, the following query returns the character string 123456:

```
SELECT '123' + '456'
```
You can use the CAST or CONVERT functions to explicitly convert data types.

**Bitwise operators**

Bitwise operators perform bit manipulations between two expressions. The following operators can be used on integer data types in UltraLite.

<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. The bitwise operators operate on integer values using the bit representation of the values.

**Example**

The following statement selects rows in which the specified bits are set.

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

**Operator precedence**

The precedence of operators in expressions is as follows. Expressions in parentheses are evaluated first, then multiplication and division before addition and subtraction. String concatenation happens after addition and subtraction. The operators at the top of the list are evaluated before those at the bottom of the list.

**Tip**

Make the order of operation explicit in UltraLite, rather than relying on an operator precedence. That means, when you use more than one operator in an expression you should order operations clearly with parentheses.

1. names, functions, constants, IF expressions, CASE expressions
2. ()
3. unary operators (operators that require a single operand): +, -
4. ~
5. & , | , ^
6. * , / , %
7. + , -
8. ||
10. comparisons: IS [NOT] TRUE, FALSE, UNKNOWN
11. NOT
12. AND
13. OR

Variables in UltraLite

You cannot use SQL variables (including global variables) in UltraLite applications.

UltraLite SQL data types

UltraLite supports a subset of the data types available in SQL Anywhere.

Character data types

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

UltraLite supports the CHAR, VARCHAR, and LONG VARCHAR data types, which are stored in a single- or multi- byte character set, and are often chosen to correspond most closely to the primary language or languages stored in the database.

Storage

Fixed character types, such as VARCHAR, are embedded in the row whereas long character types, such as LONG VARCHAR, are stored separately.

Consider your page size when creating a table with many columns of large fixed types. A full row must fit on a page, and fixed character column types are stored with a row. For example, a database created with a page size of 1000 is not able to hold character values larger than 1000 because they cannot fit on the page.

See also

- “CREATE TABLE statement [UltraLite]” on page 411
CHAR data type

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

Syntax

CHAR [ ( max-length ) ]

Remarks

CHAR is a domain, implemented as VARCHAR.

See also

- “VARCHAR data type” on page 276

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 ($2^{31} - 1$).

Indexes cannot be created on a LONG VARCHAR type.

A LONG VARCHAR type can only be used in the LENGTH and CAST functions.

See also

- “CHAR data type” on page 276
- “VARCHAR data type” on page 276
- “LENGTH function [String]” on page 341
- “CAST function [Data type conversion]” on page 308

VARCHAR data type

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

Syntax

VARCHAR [ ( max-length ) ]

Parameters

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

UltraLite databases only support byte-length semantics. A non-English character can require up to 3 bytes of storage.
UltraLite Java edition databases only support character-length semantics. `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.

UltraLite compacts data as much as possible. When a VARCHAR value does not require the number of bytes specified by `max-length`, then only the number of bytes needed to store the value is used.

**Caution**
Although it is possible to create a table with a VARCHAR column where the `max-length` exceeds the page size, an error occurs if you insert a value with a length exceeding that page size.

When evaluating expressions, the maximum length for a temporary character value is 2048 bytes.

**See also**
- “CHAR data type” on page 276
- “LONG VARCHAR data type” on page 276

**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**

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.

The FLOAT (`p`) data type is a synonym for REAL or DOUBLE, depending on the value of `p`.

For information about changing the defaults by setting database options, see “UltraLite precision creation parameter” on page 146 and “UltraLite scale creation parameter” on page 147.

**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 278
- “INTEGER data type” on page 281
- “SMALLINT data type” on page 283
- “TINYINT data type” on page 283
- “UltraLite numeric functions” on page 298
- “UltraLite aggregate functions” on page 294

**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 bit 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.
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 the appropriate creation parameter.

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 + INT(((precision - scale) + 1) / 2) + INT((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.

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.

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.
DOUBLE data type

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

Syntax

```sql
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 280
- “REAL data type” on page 282
- “DOUBLE data type” on page 280
- “NUMERIC data type” on page 282
- “UltraLite numeric functions” on page 298
- “UltraLite aggregate functions” on page 294
- “UltraLite precision creation parameter” on page 146
- “UltraLite scale creation parameter” on page 147

FLOAT data type

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

Syntax

```sql
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 280
- “REAL data type” on page 282
- “DECIMAL data type” on page 279
- “NUMERIC data type” on page 282
- “UltraLite numeric functions” on page 298
- “UltraLite aggregate functions” on page 294

**INTEGER data type**

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

Syntax

```
[ 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 \(-2^{31} \text{ to } 2^{31} - 1\), or \(-2147483648 \text{ to } 2147483647\).

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

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 277
- “BIT data type” on page 278
- “SMALLINT data type” on page 283
- “TINYINT data type” on page 283
- “UltraLite numeric functions” on page 298
- “UltraLite aggregate functions” on page 294

**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 ] ) ]
```

**Remark**

NUMERIC is a domain, implemented as DECIMAL.

See also

- “DECIMAL data type” on page 279

**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 280
- “FLOAT data type” on page 280
- “DECIMAL data type” on page 279
- “NUMERIC data type” on page 282
- “UltraLite numeric functions” on page 298
- “UltraLite aggregate functions” on page 294

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 277
- “BIT data type” on page 278
- “INTEGER data type” on page 281
- “TINYINT data type” on page 283
- “UltraLite numeric functions” on page 298
- “UltraLite aggregate functions” on page 294

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, 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 277
- “BIT data type” on page 278
- “INTEGER data type” on page 281
- “SMALLINT data type” on page 283
- “UltraLite numeric functions” on page 298
- “UltraLite aggregate functions” on page 294

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.

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 creation parameter. 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 creation parameter.

A DATE value requires 4 bytes of storage.

See also

- “CURRENT TIME special value” on page 254
- “CURRENT TIMESTAMP special value” on page 255
- “UltraLite date and time functions” on page 295
- “Date formats” [SQL Anywhere Server - SQL Reference]
- “DATE function [Date and time]” on page 320
- “UltraLite date_format creation parameter” on page 136
- “UltraLite Java edition date_format option” on page 229
- “UltraLite date_order creation parameter” on page 138
- “UltraLite Java edition date_order option” on page 230
- “DATETIME data type” on page 285
- “DATETIME function [Date and time]” on page 325
- “ISDATE function [Data type conversion]” on page 339
- “UltraLite nearest_century creation parameter” on page 142
- “UltraLite Java edition nearest_century option” on page 230
- “NOW function [Date and time]” on page 355
- “TIME data type” on page 285
- “TIMESTAMP data type” on page 286
- “CURRENT TIMESTAMP special value” on page 255
- “TIMESTAMP WITH TIME ZONE data type” on page 287

DATETIME data type

DATETIME stores date and time of day information.

Syntax

DATETIME

Remarks

DATETIME is a domain, implemented as TIMESTAMP.

See also

- “TIMESTAMP data type” on page 286

TIME data type

The TIME data type stores the time of day, containing the hour, minute, second, and fraction of a second.
Syntax

TIME

Remarks

The format in which TIME values are retrieved as strings by applications is controlled by the time_format creation parameter. 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 creation parameter.

A TIME value requires 8 bytes of storage.

See also

- “Time formats” [SQL Anywhere Server - SQL Reference]
- “CURRENT TIME special value” on page 254
- “CURRENT TIMESTAMP special value” on page 255
- “CURRENT UTC TIMESTAMP special value” on page 255
- “UltraLite date and time functions” on page 295
- “DATE data type” on page 284
- “DATETIME data type” on page 285
- “DATE function [Date and time]” on page 320
- “DATETIME function [Date and time]” on page 325
- “Expressions in UltraLite” on page 256
- “GETDATE function [Date and time]” on page 333
- “ISDATE function [Data type conversion]” on page 339
- “NOW function [Date and time]” on page 355
- “TIMESTAMP data type” on page 286
- “TIMESTAMP WITH TIME ZONE data type” on page 287
- “UltraLite timestamp_format creation parameter” on page 150
- “UltraLite Java edition timestamp_format option” on page 232
- “UltraLite timestamp_with_time_zone_format creation parameter” on page 153
- “UltraLite Java edition timestamp_with_time_zone_format option” on page 233

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 creation parameter. 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 creation parameter.

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.

When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the local timezone offset on the system is used in the final result.

See also

- “Ways to send dates and times to the database” [SQL Anywhere Server - SQL Reference]
- “CURRENT TIME special value” on page 254
- “CURRENT TIMESTAMP special value” on page 255
- “CURRENT UTC TIMESTAMP special value” on page 255
- “UltraLite date and time functions” on page 295
- “DATE data type” on page 284
- “DATETIME data type” on page 285
- “DATE function [Date and time]” on page 320
- “DATETIME function [Date and time]” on page 325
- “UltraLite date_order creation parameter” on page 138
- “UltraLite Java edition date_order option” on page 230
- “Expressions in UltraLite” on page 256
- “GETDATE function [Date and time]” on page 333
- “ISDATE function [Data type conversion]” on page 339
- “UltraLite nearest_century creation parameter” on page 142
- “UltraLite Java edition nearest_century option” on page 230
- “NOW function [Date and time]” on page 355
- “TIME data type” on page 285
- “TIMESTAMP data type” on page 286
- “TIMESTAMP WITH TIME ZONE data type” on page 287
- “UltraLite timestamp_format creation parameter” on page 150
- “UltraLite Java edition timestamp_format option” on page 232
- “UltraLite timestamp_with_time_zone_format creation parameter” on page 153
- “UltraLite Java edition timestamp_with_time_zone_format option” on page 233

**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 creation parameter. 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 creation parameter.

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.

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
SELECT 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 'Yes'
ELSE '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 TIMESTAPM 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.

- **Converting to or from TIMESTAMP WITH TIME ZONE**  
  When a TIMESTAMP value is converted to TIMESTAMP WITH TIME ZONE, the client's time zone 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” [SQL Anywhere Server - SQL Reference]
- “Ways to send dates and times to the database” [SQL Anywhere Server - SQL Reference]
- “CURRENT TIME special value” on page 254
- “CURRENT TIMESTAMP special value” on page 255
- “CURRENT UTC TIMESTAMP special value” on page 255
- “UltraLite date and time functions” on page 295
- “DATE data type” on page 284
- “DATETIME data type” on page 285
- “DATE function [Date and time]” on page 320
- “DATETIME function [Date and time]” on page 325
- “UltraLite date_order creation parameter” on page 138
- “UltraLite Java edition date_order option” on page 230
- “Expressions in UltraLite” on page 256
- “GETDATE function [Date and time]” on page 333
- “ISDATE function [Data type conversion]” on page 339
- “UltraLite nearest_century creation parameter” on page 142
- “UltraLite Java edition nearest_century option” on page 230
- “NOW function [Date and time]” on page 355
- “TIME data type” on page 285
- “TIMESTAMP data type” on page 286
- “TIMESTAMP WITH TIME ZONE data type” on page 287
- “UltraLite timestamp_format creation parameter” on page 150
- “UltraLite Java edition timestamp_format option” on page 232
- “UltraLite timestamp_with_time_zone_format creation parameter” on page 153
- “UltraLite Java edition timestamp_with_time_zone_format option” on page 233

### 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 ) ]
```

**Remarks**

BINARY is a domain, implemented as VARBINARY.

**See also**

- “VARBINARY data type” on page 291
LONG BINARY data type

The LONG BINARY data type stores binary data of arbitrary length.

Syntax

```sql
LONG BINARY
```

Remarks

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

Indexes cannot be created on a LONG BINARY type.

A LONG BINARY type can only be used in the LENGTH and CAST functions.

See also

- “BINARY data type” on page 289
- “VARBINARY data type” on page 291

UNIQUEIDENTIFIER data type

The UNIQUEIDENTIFIER data type stores UUID (also known as GUID) values.

Syntax

```sql
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 setting by using the UUIDTOSTR and STRTOUUID functions.

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.
See also

- “NEWID function [Miscellaneous]” on page 354
- “UUIDTOSTR function [String]” on page 391
- “STRTOUUID function [String]” on page 382
- “UltraLite string functions” on page 299

**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 compared to the other as though the shorter string were padded with zeros.

When evaluating expressions, the maximum length for a temporary character value is 2048 bytes.

VARBINARY values are not transformed during character set conversion.

See also

- “BINARY data type” on page 289
- “LONG BINARY data type” [SQL Anywhere Server - SQL Reference]
- “UltraLite string functions” on page 299
- “Bitwise operators” on page 274

**Spatial data types**

**Spatial data** is data that describes the position, shape, and orientation of objects in a defined space. UltraLite provides storage and data management features for spatial data, in the form of points, allowing you to store information such as geographic locations and routing information. Points are defined using a **spatial type**, ST_GEOMETRY. You use functions and constructors to access and manipulate the spatial data. UltraLite also provides a set of SQL spatial functions designed for compatibility with other products.

A point defines a single location in space. A point geometry does not have length or area. A point always has an X and Y coordinate.
In GIS data, points are typically used to represent locations such as addresses, or geographic features such as a mountain.

See also

- “Recommended reading on spatial topics” [SQL Anywhere Server - Spatial Data Support]

**ST_GEOMETRY data type**

The ST_GEOMETRY type is used to store spatial data in the form of points.

**Functions**

UltraLite supports the following functions of ST_GEOMETRY:

- “ST_AsBinary function [Spatial]”
- “ST_AsExtText function [Spatial]”
- “ST_AsText function [Spatial]”
- “ST_Distance function [Spatial]”
- “ST_Equals function [Spatial]”
- “ST_IntersectsRect function [Spatial]”
- “ST_Point function [Spatial]”
- “ST_PointFromExtText function [Spatial]”
- “ST_PointFromText function [Spatial]”
- “ST_PointFromWKB function [Spatial]”
- “ST_SRID function [Spatial]”
- “ST_X function [Spatial]”
- “ST_Y function [Spatial]”

**Remarks**

The ST_GEOMETRY type is the maximal supertype of the geometry type hierarchy. The ST_GEOMETRY type supports methods that can be applied to any spatial value. The ST_GEOMETRY type cannot be instantiated; instead, a subtype should be instantiated. When working with original formats (WKT or WKB), you can use methods such as ST_PointFromText/ST_PointFromWKB to instantiate the appropriate concrete type representing the value in the original format.

The ST_SRID method can be used to retrieve the spatial reference system associated with the value.

Columns of the ST_GEOMETRY type or any of its subtypes cannot be included in a primary key, unique index, or unique constraint.

**Column and object definitions**

UltraLite provides a fixed set of three different reference systems that can be attributed to a column during its creation. Individual geometry objects can be associated with any SRID value except the undefined reference system, and can only be stored in a column associated with a matching SRID value or the undefined reference system.

The predefined reference systems are:
• **Undefined or "null" reference system**  This is the default reference system if no SRID value is provided. It allows contained geometry values to be in any valid reference system. This reference system allows for catch-all columns that do not enforce any reference system consistency among their geometry objects.

• **Default planar reference system**  Defined by specifying a SRID value of 0 during column creation, this column can contain only geometry values associated with this reference system. The values are treated as being in 2D planar space.

• **WGS 84 Geodetic Reference System**  Defined by specifying a SRID value of 4326 during column creation, this column can only contain geometry values associated with this reference system. The values are treated as being on the Earth's surface and operations are applied accordingly.

**Note**
A point in SRID 4326 can be stored in a column with the WGS 84 reference system or with the undefined reference system, but not in the default planar system.

No transformations between reference systems are supported.

**Example**

The following example illustrates how to create a table with one column associated with the default planar reference system and one with an undefined reference system:

```
CREATE TABLE T1 (  
  V1 INTEGER PRIMARY KEY,  
  V2 ST_GEOMETRY(SRID=0),  
  V3 ST_GEOMETRY
)
```

The following SQL statement illustrates how to insert data into the T1 table from the previous example:

```
INSERT INTO T1(V1, V2, V3)  
VALUES (1, ST_POINTFROMTEXT('POINT(10 20)', 0), ST_POINT(5, 6, 2163))
```

**User-defined data types and their equivalents**

Unlike SQL Anywhere databases, UltraLite does not support user-defined data types. The following table lists UltraLite data type equivalents to built-in SQL Anywhere aliases:

<table>
<thead>
<tr>
<th>SQL Anywhere data type</th>
<th>UltraLite equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONEY</td>
<td>DECIMAL(19,4)</td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>DECIMAL(10,4)</td>
</tr>
<tr>
<td>TEXT</td>
<td>LONG VARCHAR</td>
</tr>
<tr>
<td>XML</td>
<td>LONG VARCHAR</td>
</tr>
</tbody>
</table>
See also
  ● “LONG VARCHAR data type” on page 276
  ● “DECIMAL data type” on page 279

UltraLite SQL functions

Functions are used to return information from the database. They are allowed anywhere an expression is allowed.

Unless otherwise specified in the documentation, NULL is returned for a function if any argument is NULL.

Functions use the same syntax conventions used by SQL statements.

See also
  ● “Syntax conventions” [SQL Anywhere Server - SQL Reference]

Function types

This section groups the available function by type.

UltraLite supports a subset of the same functions documented for SQL Anywhere, and sometimes with a few differences.

Note
Unless otherwise stated, any function that receives NULL as a parameter returns NULL.

See also
  ● “ST_GEOMETRY data type” on page 292

UltraLite 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:

- “AVG function [Aggregate]”
- “COUNT function [Aggregate]”
- “COUNT_UPLOAD_ROWS function [Aggregate]”
- “LIST function [Aggregate]”
- “MAX function [Aggregate]”
- “MIN function [Aggregate]”
- “SUM function [Aggregate]”

UltraLite 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:

- “CAST function [Data type conversion]”
- “CONVERT function [Data type conversion]”
- “HEXTOINT function [Data type conversion]”
- “INTTOHEX function [Data type conversion]”
- “ISDATE function [Data type conversion]”

UltraLite 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:

```sql
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>Date part</td>
<td>Abbreviation</td>
<td>Values</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
<td>------------------------------------------------------------------------</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.</td>
</tr>
<tr>
<td>Caldayofweek</td>
<td>CDW</td>
<td>1-7. (Monday = 1, ..., Sunday = 7)</td>
</tr>
<tr>
<td>TZOffset</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]”
- “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” [SQL Anywhere Server - SQL Reference]
- “UltraLite SQL data types” on page 275

UltraLite 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:

- “ARGN function [Miscellaneous]”
- “COALESCE function [Miscellaneous]”
- “EXPLANATION function [Miscellaneous]”
- “GREATER function [Miscellaneous]”
- “IFNULL function [Miscellaneous]”
- “ISNULL function [Miscellaneous]”
- “LESSER function [Miscellaneous]”
- “NEWID function [Miscellaneous]”
- “NULLIF function [Miscellaneous]”

UltraLite 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]”
- “REMAINDER function [Numeric]”
- “ROUND function [Numeric]”
- “SIGN function [Numeric]”
- “SIN function [Numeric]”
- “SQRT function [Numeric]”
- “TAN function [Numeric]”
- “TRUNCNUM function [Numeric]”
UltraLite spatial functions

Spatial data is data that describes the position, shape, and orientation of objects in a defined space.
UltraLite provides storage and data management features for spatial data, in the form of points, allowing
you to store information such as geographic locations and routing information, for instance. UltraLite
provides a set of SQL spatial functions designed for compatibility with other products. You use these
functions and constructors to access and manipulate the spatial data.

List of functions

The following spatial functions are available:

- “ST_AsBinary function [Spatial]”
- “ST_AsExtText function [Spatial]”
- “ST_AsText function [Spatial]”
- “ST_Distance function [Spatial]”
- “ST_Equals function [Spatial]”
- “ST_IntersectsRect function [Spatial]”
- “ST_Point function [Spatial]”
- “ST_PointFromExtText function [Spatial]”
- “ST_PointFromText function [Spatial]”
- “ST_PointFromWKB function [Spatial]”
- “ST_SRID function [Spatial]”
- “ST_X function [Spatial]”
- “ST_Y function [Spatial]”

UltraLite 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]”
- “BYTE_LENGTH function [String]”
- “BYTE_SUBSTR function [String]”
- “CHAR function [String]”
- “CHARINDEX function [String]”
- “CHAR_LENGTH function [String]”
- “DIFFERENCE function [String]”
- “INSERTSTR function [String]”
- “LCASE function [String]”
- “LEFT function [String]”
- “LENGTH function [String]”
- “LOCATE function [String]”
- “LOWER function [String]”
- “LTRIM function [String]”
- “PATINDEX function [String]”
- “REPEAT function [String]”
- “REPLACE function [String]”
- “REPLICATE function [String]”
- “RIGHT function [String]”
- “RTRIM function [String]”
- “SIMILAR function [String]”
- “SOUNDEX function [String]”
- “SPACE function [String]”
- “STR function [String]”
- “STRING function [String]”
- “STRTOUUID function [String]”
- “STUFF function [String]”
- “SUBSTRING function [String]”
- “TRIM function [String]”
- “UCASE function [String]”
- “UPPER function [String]”
- “UUIDTOSTR function [String]”

UltraLite system functions

System functions return system information.

List of functions

The following system functions are available in UltraLite:

- “DB_PROPERTY function [System]”
- “ML_GET_SERVER_NOTIFICATION function [System]”
- “SYNC_PROFILE_OPTION_VALUE function [System]”
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 294.

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>

Example
The following statement returns the value 66:

```
SELECT ABS( -66 );
```

ACOS function [Numeric]
Returns the arc-cosine, in radians, of a numeric expression. Not supported by UltraLite Java edition databases.

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 303
    ● “ATAN function [Numeric]” on page 304
    ● “ATAN2 function [Numeric]” on page 305
    ● “COS function [Numeric]” on page 316

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.

Example

    The following statement returns the value 6:

        SELECT ARGN( 6, 1, 2, 3, 4, 5, 6 );
ASCII function [String]

Returns the integer ASCII value of the first byte in a string-expression.

Syntax

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 311
- “UltraLite string functions” on page 299

Example

The following statement returns the value 90:

SELECT ASCII( 'Z' );

ASIN function [Numeric]

Returns the arc-sine, in radians, of a number.

Syntax

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 301
- “ATAN function [Numeric]” on page 304
- “ATAN2 function [Numeric]” on page 305
- “SIN function [Numeric]” on page 371

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. Not supported by UltraLite Java edition databases.

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.

See also

- “ACOS function [Numeric]” on page 301
- “ASIN function [Numeric]” on page 303
- “ATAN2 function [Numeric]” on page 305
- “TAN function [Numeric]” on page 386

Example

The following statement returns the arc-tangent value for 0.52:

```
SELECT ATAN( 0.52 );
```
**ATAN2 function [Numeric]**

Returns the arc-tangent, in radians, of the ratio of two numbers. Not supported by UltraLite Java edition databases.

**Syntax**

```
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 301
- “ASIN function [Numeric]” on page 303
- “ATAN function [Numeric]” on page 304
- “TAN function [Numeric]” on page 386

**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**

```
AVG( [ DISTINCT ] numeric-expression )
```

**Parameters**

- `[ ALL ] numeric-expression` The expression whose average is calculated over the rows in each group.
- **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 numeric-expression is the NULL value.

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 also
● “SUM function [Aggregate]” on page 384
● “COUNT function [Aggregate]” on page 317

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;

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.
See also

- “CHAR_LENGTH function [String]” on page 312
- “DATALENGTH function [System]” on page 319
- “LENGTH function [String]” on page 341
- “UltraLite string functions” on page 299

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, from the left of the starting position.

**Returns**

BINARY, or VARCHAR. 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 383
- “UltraLite string functions” on page 299
Example

The following statement returns the value Test:

```sql
SELECT BYTE_SUBSTR( 'Test Message', 1, 4 );
```

**CAST function [Data type conversion]**

Returns the value of an expression converted to a supplied data type.

**Syntax**

```sql
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, a default value is selected. It is recommended that you explicitly indicate the precision and scale in your CAST function.

The ability to convert depends on the value used in the conversion. The values in the original data type must be compatible with the new data type to avoid generating a conversion error.

Use the following chart to determine whether a conversion is supported:
<table>
<thead>
<tr>
<th>FROM:</th>
<th>BIT</th>
<th>SMALLINT</th>
<th>UNSIGNED SMALLINT</th>
<th>TINYINT</th>
<th>UNSIGNED INTEGER</th>
<th>INTEGER</th>
<th>UNSIGNED BIGINT</th>
<th>BIGINT</th>
<th>FLOAT</th>
<th>REAL</th>
<th>DOUBLE</th>
<th>NUMERIC OR DECIMAL</th>
<th>DATE</th>
<th>TIME</th>
<th>DATETIME OR TIMESTAMP</th>
<th>TIMESTAMP WITH TIME-ZONE</th>
<th>UNIQUE IDENTIFIER</th>
<th>BINARY OR VARBINARY</th>
<th>LONG BINARY</th>
<th>CHAR OR VARCHAR</th>
<th>LONG VARCHAR</th>
<th>ST_GEOGRAPHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TINYINT</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SMALLINT</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>UNSIGNED INTEGER</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>INTEGER</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BIGINT</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DOUBLE</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NUMERIC OR DECIMAL</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DATE</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TIME</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DATETIME OR TIMESTAMP</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME-ZONE</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>UNIQUE IDENTIFIER</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BINARY OR VARBINARY</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LONG BINARY</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CHAR OR VARCHAR</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ST_GEOGRAPHY</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>Always converts</td>
</tr>
<tr>
<td>✗</td>
<td>Never converts</td>
</tr>
<tr>
<td>⚠</td>
<td>Value-dependent</td>
</tr>
</tbody>
</table>
Note
To convert between a VARBINARY and a UNIQUEIDENTIFIER, the VARBINARY value must have a 16 byte length.

To convert between a NUMERIC and a VARBINARY, the NUMERIC source must have a value that can also be cast as a BIGINT.

To convert from a VARCHAR to an ST_GEOMETRY, the VARCHAR source must represent a valid geometry in either WKT or EWKT format.

To convert from a VARBINARY to an ST_GEOMETRY, the VARBINARY source must represent a valid geometry in WKB format.

When casting from a WKB or WKT formatted source to an ST_GEOMETRY, an SRID of 0 is assigned to the ST_GEOMETRY value. When casting from an ST_GEOMETRY, VARCHAR values are formatted in EWKT and VARBINARY values are formatted in WKB.

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” [SQL Anywhere Server - SQL Reference].

See also
● “CONVERT function [Data type conversion]” on page 314

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 );
```

You can use the CAST function to shorten strings:

```
SELECT CAST ( 'Surname' AS CHAR(5) );
```

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.

See also

- “FLOOR function [Numeric]” on page 332

Example

The following statement returns the value 60:

```
SELECT CEILING( 59.84567 )
```

**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

- “UltraLite string functions” on page 299
Example

The following statement returns the value 8:

```
SELECT CHAR( 89 );
```

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, and LONG VARCHAR data types. However, you must use the LENGTH function for BINARY and bit array data types.

See also

- “BYTE_LENGTH function [String]” on page 306
- “UltraLite string functions” on page 299

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.

See also

- “SUBSTRING function [String]” on page 383
- “REPLACE function [String]” on page 363
- “LOCATE function [String]” on page 343
- “UltraLite string functions” on page 299

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

```sql
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 339.

See also

- “ISNULL function [Miscellaneous]” on page 339

Example

The following statement returns the value 34:

```sql
SELECT COALESCE( NULL, 34, 13, 0 );
```

**CONVERT function [Data type conversion]**

Returns an expression converted to a supplied data type.

This function is similar to the CAST function but allows you to specify a format style to assist with date and time data type conversions. For more information about other conversions, see “CAST function [Data type conversion]” on page 308.

Syntax

```sql
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 (yyyyMMdd) style codes</th>
<th>Output format</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0 or 100</td>
<td>MMM dd yyyy hh:mm:ss</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/yyyy[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:mm:ss</td>
</tr>
<tr>
<td>-</td>
<td>9 or 109</td>
<td>MMM dd yyyy hh:mm:ss:sss</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]yyyyyymmdd</td>
</tr>
<tr>
<td>-</td>
<td>13 or 113</td>
<td>dd MMM yyyy hh:mm:ss:sss (24 hour clock, Europe default + milliseconds, 4-digit year)</td>
</tr>
<tr>
<td>-</td>
<td>14 or 114</td>
<td>hh:mm:ss:sss (24 hour clock)</td>
</tr>
<tr>
<td>-</td>
<td>20 or 120</td>
<td>yyyy-mm-dd hh:mm:ss (24-hour clock, ODBC canonical, 4-digit year)</td>
</tr>
<tr>
<td>-</td>
<td>21 or 121</td>
<td>yyyy-mm-dd hh:mm: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 “UltraLite date_format creation parameter” on page 136.

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>
</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>
<tr>
<td>Mar 23, 00</td>
</tr>
<tr>
<td>Mar 25, 00</td>
</tr>
</tbody>
</table>

The following statement illustrates conversion to an integer and returns the value 5:

```
SELECT CONVERT( integer, 5.2 );
```

### 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 301
- “COT function [Numeric]” on page 317
- “SIN function [Numeric]” on page 371
- “TAN function [Numeric]” on page 386

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 316
- “SIN function [Numeric]” on page 371
- “TAN function [Numeric]” on page 386

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([ * | [ DISTINCT ] expression ])
```

Parameters

- `*` Return the number of rows in each group. COUNT(*) and COUNT() are semantically equivalent.
- `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.

See also

- “AVG function [Aggregate]” on page 305
- “SUM function [Aggregate]” on page 384
- “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:

```
SELECT City, COUNT( * ) FROM GROUPO.Employees GROUP BY City;
```

**COUNT_UPLOAD_ROWS function [Aggregate]**

Returns a count of the number of rows that will be uploaded in the next synchronization.

Syntax

```
COUNT.Upload.Rows( pubs,threshold )
```

Parameters

- `pubs` A comma-separated list of publications to check for rows.
- `threshold` The maximum number of rows to count (a value of 0 corresponds to the maximum limit).

Returns

INT
Example
The following returns the total number of rows to upload in mypub1 and mypub2:

```
SELECT COUNT_UPLOAD_ROWS('mypub1,mypub2', 0);
```

**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>
<tr>
<td>TIME</td>
<td>8</td>
</tr>
<tr>
<td>DATE</td>
<td>4</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>8</td>
</tr>
<tr>
<td>DATETIME</td>
<td>8</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>29</td>
</tr>
<tr>
<td>Data type</td>
<td>DATALENGTH</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>16</td>
</tr>
<tr>
<td>CHAR</td>
<td>Length of the data</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>Length of the data</td>
</tr>
<tr>
<td>BINARY</td>
<td>Length of the data</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>Length of the data</td>
</tr>
<tr>
<td>NCHAR</td>
<td>Length of the data</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>Length of the data</td>
</tr>
<tr>
<td>TEXT</td>
<td>Length of the data</td>
</tr>
<tr>
<td>NTEXT</td>
<td>Length of the data</td>
</tr>
<tr>
<td>IMAGE</td>
<td>Length of the data</td>
</tr>
<tr>
<td>XML</td>
<td>Length of the data</td>
</tr>
</tbody>
</table>

See also

- “SQL data types” [SQL Anywhere Server - SQL Reference]

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 ‘8sdofinsv8s7a7s7gehe4h’:

```sql
SELECT DATALENGTH( '8sdofinsv8s7a7s7gehe4h' );
```

**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 “UltraLite date_order creation parameter” on page 138.

**Syntax**

```
DATE( expression )
```

**Returns**

```
DATE
```
Parameters

- **expression**  The value to be converted to date format, typically a string.

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**  The date part that `integer-expression` represents.
  
  For a complete listing of allowed date parts, see “Specifying date parts” on page 295.

- **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.

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:

SELECT DATEADD( day, 4, CAST( '1987/05/02 11:33:00.000000+04:00' as TIMESTAMP WITH TIME ZONE ));

**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 295.

- **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.

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 “UltraLite date_format creation parameter” on page 136.

**Returns**

VARCHAR

**Remarks**

Any allowable date format can be used for the string-expression.

**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 TIMESTAMP 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 295.
  
- **timestamp-expression** The TIMESTAMP or TIMESTAMP WITH TIME ZONE value for which the date part name is to be returned. For meaningful results, *timestamp-expression* 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.

**See also**

- “DATEPART function [Date and time]” on page 325

**Example**

The following statement returns the value May:

```
SELECT DATENAME( month, '1987/05/02' );
```
DATEPART function [Date and time]

Syntax

\[
\text{DATEPART( } \text{date-part, timestamp-expression } \text{)}
\]

Parameters

- **date-part**  The date part to be returned.
  
  For a complete listing of allowed date parts, see “Specifying date parts” on page 295.

- **timestamp-expression**  The TIMESTAMP or TIMESTAMP WITH TIME ZONE value for which the part is to be returned.

Returns

\[
\text{INT}
\]

Remarks

For meaningful results `timestamp-expression` should contain the required `date-part` portion.

Example

The following statement returns the value 5:

\[
\text{SELECT DATEPART( month, '1987/05/02' );}
\]

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:

\[
\begin{align*}
\text{CREATE TABLE TableStatistics (} \\
\quad \text{ID INTEGER NOT NULL DEFAULT AUTOINCREMENT,} \\
\quad \text{Year INT,} \\
\quad \text{NumberOrders INT);} \\
\text{INSERT INTO TableStatistics (Year, NumberOrders) } \\
\text{SELECT DATEPART( Year, OrderDate ), COUNT(*)} \\
\text{FROM GROUPO.SalesOrders} \\
\text{GROUP BY DATEPART( Year, OrderDate );}
\end{align*}
\]

DATETIME function [Date and time]

Converts an expression into a TIMESTAMP value.

Syntax

\[
\text{DATETIME( expression )}
\]

Parameters

- **expression**  The expression to be converted. It is generally a string.

Returns

\[
\text{TIMESTAMP}
\]

Remarks
Attempts to convert numerical values return an error.

See also
- “CAST function [Data type conversion]” on page 308

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**
```
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.

**Example**
The following statement returns the value 12:

```
SELECT DAY('2001-09-12');
```

**DAYNAME function [Date and time]**
Returns the name of the day of the week from a date.

**Syntax**
```
DAYNAME(date-expression)
```

**Parameters**
- **date-expression** The date.

**Returns**
VARCHAR
Remarks

The English names are returned as: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday.

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 `integer-expression` 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 308.

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 322
- “DATEADD function [Date and time]” on page 321

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_PROPERTY function [System]**

Returns the value of the given property. Not supported by UltraLite Java edition databases.

Syntax

```sql
DB_PROPERTY( property-name )
```

Parameters

- **property-name**  The database property name.

Returns

VARCHAR, LONG VARCHAR

Remarks

Returns a string.
To set an option in UltraLite, use the SET OPTION statement or your component's API-specific Set Database Option method.

**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**
- “UltraLite database properties” on page 177
- “SET OPTION statement [UltraLite]” on page 431
- “ULConnection.SetDatabaseOption method [UltraLite C++]” [UltraLite - C and C++ Programming]

**Example**
The following statement returns the page size of the current database, in bytes:

```
SELECT DB_PROPERTY( 'page_size');
```

### 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.

**Example**
The following statement returns the value 29.79380534680281:

```
SELECT DEGREES( 0.52 );
```
**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 372
- “UltraLite string functions” on page 299

**Example**

The following statement returns similarity between the words test and chest:

```
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**

```
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.

Example
The following statement returns the value 5:

```
SELECT DOW( '1998-07-09' );
```

The following statement returns the value 1:

```
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:

```
SELECT DOW( StartDate ) FROM GROUPO.Employees;
```

EXP function [Numeric]
Returns the result of the base of natural logarithms e raised to the power of the given argument. Not supported by UltraLite Java edition databases.

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.

Example
The statement returns the value 3269017.3724721107:

```
SELECT EXP( 15 );
```

EXPLANATION function [Miscellaneous]
Returns the optimization strategy of a SQL statement as a plain text string.
Syntax

```
EXPLANATION(
    string-expression
    , cursor-type ]
    [, update-status ]
)
```

EXPLANATION( string-expression )

Parameters

- **string-expression**  The SQL statement, which is commonly a SELECT statement, but can also be an UPDATE, MERGE, or DELETE statement.

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” on page 446

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' );
```

FLOOR function [Numeric]

Returns the largest integer not greater than the given number.

Syntax

```
FLOOR( 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 310

Example

The following statement returns a Floor value of 123:

```
SELECT FLOOR (123);
```

The following statement returns a Floor value of 123:

```
SELECT FLOOR (123.45);
```

The following statement returns a Floor value of -124:

```
SELECT FLOOR (-123.45);
```

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

- “NOW function [Date and time]” on page 355

Example

The following statement returns the system date and time:

```
SELECT GETDATE( );
```
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 342

Example

The following statement returns the value 10:

```
SELECT GREATER( 10, 5 ) FROM dummy;
```

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” [SQL Anywhere Server - SQL Reference].

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:

```sql
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:

```sql
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.

**See also**

- “INTTOHEX function [Data type conversion]” on page 338

**Example**

The following statement returns the value 420:

```sql
SELECT HEXTOINT('1A4');
```

---

**HOUR function [Date and time]**

Returns the hour component of a TIMESTAMP value.

**Syntax**

```sql
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.

**Example**

The following statement returns the value 21:

```sql
SELECT HOUR('1998-07-09 21:12:13');
```
HOURS function [Date and time]

Manipulates a TIMESTAMP or returns the number of hours between two TIMESTAMP values. For specific details, see this function’s usage.

Syntax 1
HOURS ( timestamp-expression )

Syntax 2
HOURS ( timestamp-expression, timestamp-expression )

Syntax 3
HOURS ( time-or-timestamp-expression, integer-expression )

Parameters
- time-or-timestamp-expression  A value of type TIME or TIMESTAMP.
- timestamp-expression  A value of type TIMESTAMP.
- integer-expression  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 308.

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 322
- “DATEADD function [Date and time]” on page 321

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).

```sql
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:

```sql
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).

```sql
SELECT HOURS( CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 );
SELECT DATEADD( hour, 5, '1999-05-12 21:05:07' );
```

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.

Example

The following statement returns the value -66:

```sql
SELECT IFNULL( NULL, -66 );
```
The following statement returns NULL, because the first expression is not NULL and there is no third
expression:

```
SELECT IFNULL( -66, -66 );
```

**INSERTSTR function [String]**

Inserts a string into another string at a specified position.

**Syntax**

```
INSERTSTR( integer-expression, string-expression-1, string-expression-2 )
```

**Parameters**

- **integer-expression**  
  The position after which the string is to be inserted. Use zero to insert a string at the beginning.

- **string-expression-1**  
  The string into which the other string is to be inserted.

- **string-expression-2**  
  The string to be inserted.

**Returns**

LONG VARCHAR

**See also**

- “STUFF function [String]” on page 382
- “UltraLite string functions” on page 299

**Example**

The following statement returns the value **backoffice**:

```
SELECT INSERTSTR( 0, 'office ', 'back' );
```

**INTTOHEX function [Data type conversion]**

Returns a string containing the hexadecimal equivalent of an integer.

**Syntax**

```
INTTOHEX( integer-expression )
```

**Parameters**

- **integer-expression**  
  The integer to be converted to hexadecimal.

**Returns**

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 334
● “Converting to and from hexadecimal values” [SQL Anywhere Server - SQL Reference]

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.

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 313

Example

The following statement returns the value -66:

```
SELECT ISNULL( NULL , -66, 55, 45, NULL, 16 );
```

**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
- LONG VARCHAR
- VARCHAR

**Remarks**

The LCASE function is identical to the LOWER function.

See also

- “LOWER function [String]” on page 345
- “UCASE function [String]” on page 389
- “UPPER function [String]” on page 390
- “UltraLite string functions” on page 299

Example

The following statement returns the value chocolate:

```
SELECT LCASE( 'ChoCOlatE' );
```

**LEFT function [String]**

Returns multiple characters from the beginning of a string.

**Syntax**

```
LEFT( string-expression, integer-expression )
```
Parameters

- **string-expression**  The string.
- **integer-expression**  The number of characters to return.

Returns

LONG VARCHAR

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.

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 365
- “UltraLite string functions” on page 299

Example

The following statement returns the first 5 characters of each Surname value in the Customers table:

```
SELECT LEFT( Surname, 5) FROM GROUPO.Customers;
```

**LENGTH function [String]**

Returns the number of characters in the specified string.

Syntax

```
LENGTH( string-expression )
```

Parameters

- **string-expression**  The string.

Returns

INT

Remarks

Use this function to determine the length of a string. For example, specify a column name for string-expression to determine the length of values in the column.

If the string contains multibyte characters, and the proper collation is being used, LENGTH returns the number of characters, not the number of bytes. If the string is of data type BINARY, the LENGTH function behaves as the BYTE_LENGTH function.
Note
You can use the LENGTH function and the CHAR_LENGTH function interchangeably for CHAR, VARCHAR, and LONG VARCHAR data types. However, you must use the LENGTH function for BINARY and bit array data types.

See also
- “BYTE_LENGTH function [String]” on page 306
- “International languages and character sets” [SQL Anywhere Server - Database Administration]
- “UltraLite string functions” on page 299

Example
The following statement returns the value 9:

```sql
SELECT LENGTH( 'chocolate' );
```

LESSER function [Miscellaneous]
Returns the lesser of two parameter values.

Syntax
```
LESSER( 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 value is returned.

See also
- “GREATER function [Miscellaneous]” on page 334

Example
The following statement returns the value 5:

```sql
SELECT LESSER( 10, 5 ) FROM dummy;
```
LIST function [Aggregate]

Returns a delimited list of values for every row in a group.

Syntax

LIST(
    [ DISTINCT ] string-expression
    [ , delimiter-string ]
)

Parameters

- string-expression A string expression, usually a column name. 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.

Returns

LONG VARCHARM

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.

Examples

The following statement returns all street addresses from the Employees table:

    SELECT LIST( Street ) FROM GROUPO.Employees;

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.

- 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 \textit{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 \textit{byte} positions.

**See also**

- “UltraLite string functions” on page 299
- “CHARINDEX function [String]” on page 312

**Example**

The following statement returns the value 8:

```sql
SELECT LOCATE('office party this week - rsvp as soon as possible', 'party', 2);
```

**LOG function [Numeric]**

Returns the natural logarithm of a number. Not supported by UltraLite Java edition databases.

**Syntax**

`LOG( numeric-expression )`

**Parameters**

- \textit{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 345

Example

The following statement returns the natural logarithm of 50:

```
SELECT LOG ( 50 );
```

LOG10 function [Numeric]

Returns the base 10 logarithm of a number. Not supported by UltraLite Java edition databases.

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 344

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, or LONG VARCHAR 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 340
● “UCASE function [String]” on page 389
● “UPPER function [String]” on page 390
● “UltraLite string functions” on page 299

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
● LONG VARCHAR

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.

See also
● “RTRIM function [String]” on page 366
● “TRIM function [String]” on page 388
● “UltraLite string functions” on page 299

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( [ DISTINCT ] expression )
```

Parameters

- `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.

For simple comparisons of two expressions, you can also use the GREATER function.

See also

- “MIN function [Aggregate]” on page 347
- “Troubleshooting: Aggregate functions and outer references” [SQL Anywhere 16 - Changes and Upgrading]

Example

The following statement returns the value 138948.000, representing the maximum salary in the Employees table:

```
SELECT MAX( Salary )
FROM GROUPO.Employees;
```

MIN function [Aggregate]

Returns the minimum expression value found in each group of rows.

Syntax 1

```
MIN( [ DISTINCT ] expression )
```

Parameters

- `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.

For simple comparisons of two expressions, you can also use the LESSER function. See “LESSER function [Miscellaneous]” [SQL Anywhere Server - SQL Reference].

See also

● “MAX function [Aggregate]” on page 347

Example

The following statement returns the value 24903.000, representing the minimum salary in the Employees table:

```sql
SELECT MIN( Salary )
FROM GROUPO.Employees;
```

MINUTE function [Date and time]

Returns the minute component of a TIMESTAMP value.

Syntax

```sql
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.

Example

The following statement returns the value 22:

```sql
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.

  **Note**  
  0000-02-29 is not meant to imply an actual date; it is the default date used by the MINUTES function.

- **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]” [SQL Anywhere Server - SQL Reference]
- “DATEADD function [Date and time]” [SQL Anywhere Server - SQL Reference]
- “CAST function [Data type conversion]” on page 308

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).

```
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:

```
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).

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07' AS TIMESTAMP ), 5 );
SELECT DATEADD( minute, 5, '1999-05-12 21:05:07' );
```

ML_GET_SERVER_NOTIFICATION function [System]
This function allows UltraLite users to use light weight polling to query a notifier on a MobiLink server for server-initiated sync requests. Not supported by UltraLite Java edition databases.

Syntax
```
ML_GET_SERVER_NOTIFICATION( notifier, address, key )
```

Parameters
- **notifier** The name of the notifier on the MobiLink server to poll.
- **address** The stream parameters, specified as:
  ```
tcip(host=pc1;port=1234)
  ```
- **key** Optional. The notification request key.

Returns
Returns the subject and content of a notification request for the given request key.

Remarks
If there are no requests for the given request key, or if the notifier name does not exist on the MobiLink server, the result is NULL. If NULL is provided for the request key, then the remote ID of the user is used.
as the request key. If a request does exist, the resulting message is returned in the form: [subject]content (for example, [sync]profile1).

This function communicates over the network as it retrieves responses from the MobiLink server. As a result, this function may require a long execution time resulting from network latency. During execution, there may be periods when the function can execute in the background, allowing work to be performed in the runtime on other connections. These periods are not guaranteed however, and depend on the complexity of the SQL. The recommended method for users to retrieve a MobiLink address to use in this function is to use the sync_profile_option_value function with an existing synchronization profile to get the value for the Stream profile option. The value returned by this function call can be used directly as the MobiLink address parameter.

See also
● “SYNC_PROFILE_OPTION_VALUE function [System]” on page 386

Example
SELECT ML_GET_SERVER_NOTIFICATION('Notifier1', 'tcpip{host=sybase;port=1234}', 'MyKey');

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 362

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.

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 325

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 308.

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 322
● “DATEADD function [Date and time]” on page 321

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' );
```

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.
See also

- “The NEWID default” [SQL Anywhere Server - SQL Usage]
- “STRTOUUID function [String]” on page 382
- “UUIDTOSTR function [String]” on page 391

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.

**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**

```sql
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

- “GETDATE function [Date and time]” on page 333
- “CURRENT TIMESTAMP special value” on page 255

Example

The following statement returns the current date and time:

```sql
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 259

**Example**

The following statement returns the value `a`:

```
SELECT NULLIF( 'a', 'b' );
```

The following statement returns NULL:

```
SELECT NULLIF( 'a', 'a' );
```

**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 for UltraLite uses the following wildcards:

<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**

- “LOCATE function [String]” on page 343
- “UltraLite string functions” on page 299

**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 $$$$ jumped over the lazy dog' ),
( 'the quick brown fox 0999 jumped over the lazy dog' ),
( 'the quick brown fox ** jumped over the lazy dog' ),
( 'thequickbrownfoxjumpedoverthelazydog' ),
( 'thequickbrownfoxjum999pedoverthelazyped') ,
( 'thequick$$$$brownfox' ),
( 'the quick brown fox$$ 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-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:
    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 $$$$ jumped over the lazy dog</td>
<td>4</td>
<td>21</td>
<td></td>
<td>$$$$ jumped over the lazy dog</td>
</tr>
<tr>
<td>the quick brown fox 0999 jumped over the lazy dog</td>
<td>4</td>
<td>0</td>
<td></td>
<td>the quick brown fox 0999 jumped over the lazy dog</td>
</tr>
<tr>
<td>the quick brown fox ** jumped over the lazy dog</td>
<td>4</td>
<td>21</td>
<td></td>
<td>** jumped over the lazy dog</td>
</tr>
<tr>
<td>thequickbrownfoxjumpe- doverthelazydog</td>
<td>0</td>
<td>0</td>
<td></td>
<td>thequickbrownfoxjum- pedoverthelazydog</td>
</tr>
<tr>
<td>thequickbrownfox- jum999pedoverthelazyped</td>
<td>0</td>
<td>0</td>
<td></td>
<td>thequickbrownfox- jum999pedoverthelazy- dog</td>
</tr>
<tr>
<td>thequick$$$$brownfox</td>
<td>9</td>
<td>9</td>
<td></td>
<td>thequick$ $$$$brownfox</td>
</tr>
</tbody>
</table>
PI function [Numeric]
Returns the numeric value PI.

Syntax
PI( [ * ] )

Returns
DOUBLE

Remarks
This function returns a DOUBLE value.

PI(*) and PI() are semantically equivalent.

Example
The following statement returns the value 3.141592653(...):

```
SELECT PI( * );
```
Example
The following statement returns the value 64:

```sql
SELECT POWER( 2, 6 );
```

**QUARTER function [Date and time]**

Returns a number indicating the quarter of the year from the supplied TIMESTAMP expression.

**Syntax**

```sql
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>

Example
The following statement returns the value 2:

```sql
SELECT QUARTER( '1987/05/02' );
```

**RADIANS function [Numeric]**

Converts a number from degrees to radians.

**Syntax**

```sql
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.

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. Not supported by UltraLite Java edition databases.

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].

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);
```

**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 351
Example

The following statement returns the value 2:

```sql
SELECT REMAINDER( 5, 3 );
```

**REPEAT function [String]**

Concatenates a string a specified number of times.

**Syntax**

```sql
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

**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.

**See also**

- “REPLICATE function [String]” on page 364
- “UltraLite string functions” on page 299

**Example**

The following statement returns the value `repeatrepeatrepeat`:

```sql
SELECT REPEAT( 'repeat', 3 );
```

**REPLACE function [String]**

Replaces a string with another string, and returns the new results.

**Syntax**

```sql
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

**Remarks**
This function replaces all occurrences.

Comparisons are case-sensitive on case-sensitive databases.

**See also**
- “SUBSTRING function [String]” on page 383
- “CHARINDEX function [String]” on page 312
- “UltraLite string functions” on page 299

**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

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.

See also

● “REPEAT function [String]” on page 363
● “UltraLite string functions” on page 299

Example

The following statement returns the value repeatrepeatrepeat:

```sql
SELECT REPLICATE( 'repeat', 3 );
```

**RIGHT function [String]**

Returns the rightmost characters of a string.

Syntax

```sql
RIGHT( string-expression, integer-expression )
```

Parameters

- `string-expression`  The string to return the rightmost characters for.
- `integer-expression`  The number of characters at the end of the string to return.

Returns

LONG VARCHAR

Remarks

If the string contains multibyte characters, 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 column. In this case, the entire value is returned.

Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics.
See also

- “LEFT function [String]” on page 340
- “International languages and character sets” [SQL Anywhere Server - Database Administration]
- “UltraLite string functions” on page 299

Example

The following statement returns the last 5 characters of each Surname value in the Customers table:

```
SELECT RIGHT( Surname, 5 ) FROM GROUPO.Customers;
```

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 389

Example

The following statement returns the value 123.200:

```
SELECT ROUND( 123.234, 1 );
```

RTRIM function [String]

Removes trailing blanks from the string.

Syntax

```
RTRIM( string-expression )
```
Parameters
- **string-expression**  The string to be trimmed.

Returns
- VARCHAR
- LONG VARCHAR

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.

See also
- “TRIM function [String]” on page 388
- “LTRIM function [String]” on page 346
- “UltraLite string functions” on page 299

Example
The following statement returns the string **Test Message**, with all trailing blanks removed:

```sql
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.

Example

The following statement returns the value 25.

```sql
SELECT SECOND( '1998-07-13 21:21:25' );
```
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
SECONDS( timestamp-expression )

Syntax 2
SECONDS( timestamp-expression, timestamp-expression )

Syntax 3
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 308
- “DATEADD function [Date and time]” on page 321
- “DATEDIFF function [Date and time]” on page 322

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' );
```

**SHORT_PLAN function [Miscellaneous]**

Returns a short description of the UltraLite plan optimization strategy of a SQL statement, as a string. The description is the same as that returned by the EXPLANATION function.

**Syntax**

```
SHORT_PLAN( string-expression )
```

**Remarks**

For some queries, the execution plan for UltraLite may differ from the plan selected for SQL Anywhere.

**Parameters**

- **string-expression** The SQL statement, which is commonly a SELECT statement, but can also be an UPDATE or DELETE statement.

**Returns**

LONG VARCHAR

**See also**

- “EXPLANATION function [Miscellaneous]” on page 331
Example
The following statement passes a SELECT statement as a string parameter and returns the plan for executing the query.

```sql
SELECT SHORT_PLAN(
    'SELECT * FROM GROUPO.Departments WHERE DepartmentID > 100' );
```

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.

Example
The following statement returns the value -1

```sql
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
● “UltraLite string functions” on page 299

Example
The following statement returns the value 75, indicating that the two values are 75% similar.

```
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 303
● “COS function [Numeric]” on page 316
● “COT function [Numeric]” on page 317
● “TAN function [Numeric]” on page 386
Example

The following statement returns the SIN value of 0.52.

```
SELECT SIN( 0.52 );
```

**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**

- “UltraLite string functions” on page 299

**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
● “UltraLite string functions” on page 299

Example
The following statement returns a string containing 10 spaces.

```sql
SELECT SPACE( 10 );
```

**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.

Example
The following statement returns the value 3.

```sql
SELECT SQRT( 9 );
```

**ST_AsBinary function [Spatial]**
Returns a binary string representing the specified geometry. The output format is WKB as defined by OGC SFS 1.1. This format does not contain Z and M values.
Syntax

\texttt{ST\_AsBinary\(\text{geometry-expression}\)}

Returns

- \texttt{BINARY} \quad \text{Returns the WKB representation of the \textit{geometry-expression}.}

Example

The following statement returns the result
\begin{verbatim}
0x01010000000000000000f0000000000000040:
\end{verbatim}

\begin{verbatim}
SELECT ST\_AsBinary(ST\_Point(1.0, 2.0, 4326))
\end{verbatim}

The server implicitly invokes the \texttt{ST\_AsBinary} function when converting geometries to \texttt{BINARY}. For example, the following statement returns the result
\begin{verbatim}
0x01010000000000000000f0000000000000040:
\end{verbatim}

\begin{verbatim}
SELECT CAST(ST\_Point(1.0, 2.0, 4326) AS BINARY(50))
\end{verbatim}

\textbf{ST\_AsExtText function [Spatial]}

Returns a binary string representing the specified geometry. The output format is EWKT.

Syntax

\texttt{ST\_AsExtText\(\text{geometry-expression}\)}

Returns

- \texttt{VARCHAR} \quad \text{Returns the EWKT representation of the \textit{geometry-expression}.}

Example

The following statement returns the result \texttt{SRID=4326;Point (1 2)}, with the SRID included as a prefix:

\begin{verbatim}
SELECT ST\_AsExtText(ST\_Point(1.0, 2.0, 4326))
\end{verbatim}

The \texttt{ST\_AsExtText()} function is implicitly invoked when converting geometries to \texttt{VARCHAR} types. For example, the following statement returns the result \texttt{SRID=4326;Point (1 2)}:

\begin{verbatim}
SELECT CAST(ST\_Point(1.0, 2.0, 4326) AS VARCHAR(25))
\end{verbatim}

\textbf{ST\_AsText function [Spatial]}

Returns a binary string representing the specified geometry. The output format is WKT as defined by OGC SFS 1.1.

Syntax

\texttt{ST\_AsText\(\text{geometry-expression}\)}
Returns

- **VARCHAR**  Returns the WKT representation of the *geometry-expression*.

Example

The following statement returns the result Point (1 2):

```sql
SELECT ST_AsText(ST_Point(1.0, 2.0, 4326))
```

**ST_Distance function [Spatial]**

Returns the smallest distance between two specified geometry values. If the points are in SRID 4326, the units are in meters.

Syntax

`ST_Distance(geo1, geo2)`

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo1</td>
<td>ST_Geometry</td>
<td>The first geometry value to be used to calculate the distance between two geometry values.</td>
</tr>
<tr>
<td>geo2</td>
<td>ST_Geometry</td>
<td>The second geometry value to be used to calculate the distance between two geometry values.</td>
</tr>
</tbody>
</table>

Returns

- **DOUBLE**  Returns the smallest distance between the specified geometry values.

Example

The following statement returns the result 3367142.4632130372:

```sql
SELECT ST_Distance(ST_Point(-79.38,43.65,4326),ST_Point(-123.1,49.28,4326))
```

**ST_Equals function [Spatial]**

Tests whether an ST_Geometry value is spatially equal to another ST_Geometry value. Two geometry values can be considered equal if they have the same x and y coordinates and are in the same reference system.

The test may be limited by the resolution of the spatial reference system or the accuracy of the data.

The ST_Equals function defines the semantics used for comparison predicates (= and <>), IN list predicates, DISTINCT, and GROUP BY.

Syntax

`ST_Equals(geo1, geo2)`
Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo1</td>
<td>ST_Geometry</td>
<td>The first geometry value to be compared.</td>
</tr>
<tr>
<td>geo2</td>
<td>ST_Geometry</td>
<td>The second geometry value to be compared.</td>
</tr>
</tbody>
</table>

Returns

- **BIT**  Returns 1 if the two geometry values are spatially equal, otherwise 0.

Example

The following statement returns the result 1:

```sql
SELECT ST_Equals(ST_Point(1,1,4326), ST_Point(1,1,4326))
```

**ST_IntersectsRect function [Spatial]**

Tests if a point is located within the box defined by the two points specified as min and max.

Syntax

```sql
ST_IntersectsRect(location, min, max)
```

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>ST_Geometry</td>
<td>The point to be tested.</td>
</tr>
<tr>
<td>min</td>
<td>ST_Geometry</td>
<td>The minimum point value used to define the box.</td>
</tr>
<tr>
<td>max</td>
<td>ST_Geometry</td>
<td>The maximum point value used to define the box.</td>
</tr>
</tbody>
</table>

Returns

- **BIT**  Returns 1 if `location` intersects with the specified box, otherwise 0.

Example

The following statement returns the result 1:

```sql
SELECT ST_IntersectsRect(ST_Point(1,1,4326), ST_Point(0,0,4326), ST_Point(3,3,4326))
```

**ST_Point function [Spatial]**

Constructs a point based on x and y coordinates.
Syntax

\texttt{ST\_Point(x, y, SRID)}

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>DOUBLE</td>
<td>The x coordinate to use to construct the point.</td>
</tr>
<tr>
<td>y</td>
<td>DOUBLE</td>
<td>The y coordinate to use to construct the point.</td>
</tr>
<tr>
<td>SRID</td>
<td>INTEGER</td>
<td>The SRID value associated with the point.</td>
</tr>
</tbody>
</table>

Returns

- \texttt{ST\_Point}   Returns an \texttt{ST\_Geometry} value created from the input string.

Example

The following statement creates a point at \((10.0,20.0)\) in the 2163 reference system:

\begin{verbatim}
SELECT ST_Point(10.0,20.0,2163)
\end{verbatim}

\textbf{ST\_PointFromExtText function [Spatial]}

Returns an \texttt{ST\_Geometry} value, which is transformed from a VARCHAR value containing the EWKT representation of an \texttt{ST\_Geometry}.

Syntax

\texttt{ST\_PointFromText(ewkt)}

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ewkt</td>
<td>VARCHAR</td>
<td>The EWKT representation.</td>
</tr>
</tbody>
</table>

Returns

- \texttt{ST\_Geometry}   Returns an \texttt{ST\_Geometry} value created from the input string.

Example

The following statement returns the result \(\text{SRID}=4326; \text{Point}\ (10\ 20)\) to show that a point has been created at \((10,20)\) in the 4326 reference system:

\begin{verbatim}
SELECT ST_PointFromExtText('SRID=4326;Point(10 20)')
\end{verbatim}
### ST_PointFromText function [Spatial]

Returns an ST_Geometry value, which is transformed from a VARCHAR value containing the WKT representation of an ST_Geometry.

**Syntax**

```
ST_PointFromText(wkt, srid)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wkt</td>
<td>VARCHAR</td>
<td>The WKT representation.</td>
</tr>
<tr>
<td>srid</td>
<td>INT</td>
<td>The spatial reference system identifier of the result is indicated by the SRID parameter.</td>
</tr>
</tbody>
</table>

**Returns**

- **ST_Geometry**  
  Returns an ST_Geometry value created from the input string.

  The spatial reference system identifier of the result is the given by parameter `srid`.

**Example**

The following statement returns the result `SRID=4326;Point(10 20)` to show that a point has been created at (10, 20) in the 4326 reference system:

```
SELECT ST_PointFromText('Point(10 20)',4326)
```

### ST_PointFromWKB function [Spatial]

Returns an ST_Geometry value, which is transformed from a BINARY value containing the WKB representation of an ST_Geometry.

**Syntax**

```
ST_PointFromWKB(wkb, srid)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wkb</td>
<td>BINARY</td>
<td>The WKB representation.</td>
</tr>
<tr>
<td>srid</td>
<td>INTEGER</td>
<td>The SRID value associated with the point.</td>
</tr>
</tbody>
</table>

**Returns**

- **ST_Geometry**  
  Returns an ST_Geometry value created from the input string.
Example
The following statement returns the result \((1.0, 2.0, 4326):\)

\[
\text{SELECT ST_PointFromWKB(0x010100000000000000000040,4326)}
\]

**ST_SRID function [Spatial]**
Retrieves the spatial reference system (SRID) associated with the geometry value.

**Syntax**
\[
\text{ST_SRID(} \text{geo1, srid})
\]

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo1</td>
<td>ST_Geometry</td>
<td>The point value.</td>
</tr>
<tr>
<td>srid</td>
<td>INTEGER</td>
<td>The SRID value associated with the point.</td>
</tr>
</tbody>
</table>

**Returns**
- INT Returns the SRID of the geometry.

**Example**
The following statement returns the result \(4326):\)

\[
\text{SELECT ST_SRID( ST_Point ( 10, 20, 4326 ) );}
\]

**ST_X function [Spatial]**
Returns the x coordinate of the ST_Geometry value.

**Syntax**
\[
\text{ST_X(geo1)}
\]

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo1</td>
<td>ST_Geometry</td>
<td>The ST_Geometry value from which to determine the x coordinate.</td>
</tr>
</tbody>
</table>

**Returns**
- DOUBLE Returns the x coordinate of the ST_Geometry value.

**Example**
The following statement returns the result \(10.0):\)

\[
\text{SELECT ST_X( ST_Point ( 10, 20, 4326 ) );}
\]
SELECT ST_X(ST_Point(10.0, 20.0, 4326))

**ST_Y function [Spatial]**

Returns the y coordinate of the ST_Geometry value.

**Syntax**
```
.ST_Y(geo1)
```

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo1</td>
<td>ST_Geometry</td>
<td>The ST_Geometry value from which to determine the y coordinate.</td>
</tr>
</tbody>
</table>

**Returns**

- **DOUBLE** Returns the y coordinate of the ST_Geometry value.

**Example**

The following example returns the result 20.0:

```
SELECT ST_Y(ST_Point(10.0, 20.0, 4326))
```

**STR function [String]**

Returns the string equivalent of a number.

**Syntax**
```
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
● “UltraLite string functions” on page 299

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 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

● “UltraLite string functions” on page 299

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 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. You do not need to use STRTOUUID and UUIDTOSTR functions with these versions.

For more information, see “UltraLite SQL data types” on page 275.

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.

This function is useful for inserting UUID values into a database.

See also

- “UUIDTOSTR function [String]” on page 391
- “NEWID function [Miscellaneous]” on page 354
- “UltraLite string functions” on page 299

STUFF function [String]

Deletes multiple characters from one string and replaces them with another string.

Syntax

```
STUFF( string-expression-1, start, length, string-expression-2 )
```

Parameters

- `string-expression-1` The string to be modified by the STUFF function.
- `start` The character position at which to begin deleting characters. The first character in the string is position 1.
- `length` The number of characters to delete.
The number of characters to delete.

- **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.

**Returns**

VARCHAR

See also

- “INSERTSTR function [String]” on page 338
- “UltraLite string functions” on page 299

**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

**Remarks**

In UltraLite, the database does not have an ansi_substring option, but the SUBSTR function behaves as if ansi_substring is set to on by default. The function's behavior corresponds to ANSI/ISO SQL/2008 behavior:

- **Start value** 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.
- **Length value** A positive length specifies that the substring ends length characters to the right of the starting position.
A negative \textit{length} returns an error.

A \textit{length} of zero returns an empty string.

If \textit{string-expression} is of binary data type, the \texttt{SUBSTRING} function behaves as \texttt{BYTE_SUBSTR}.

To obtain characters at the end of a string, use the \texttt{RIGHT} function.

Whenever possible, if the input string uses character-length semantics, the return value is described in character-length semantics.

\textbf{See also}

- “\texttt{BYTE_SUBSTR} function [String]” on page 307
- “\texttt{LEFT} function [String]” on page 340
- “\texttt{RIGHT} function [String]” on page 365
- “\texttt{CHARINDEX} function [String]” on page 312
- “UltraLite string functions” on page 299

\textbf{Example}

The following table shows the values returned by the \texttt{SUBSTRING} function.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{SUBSTRING( 'front yard', 1, 4 )}</td>
<td>fron</td>
</tr>
<tr>
<td>\texttt{SUBSTRING( 'back yard', 6, 4 )}</td>
<td>yard</td>
</tr>
<tr>
<td>\texttt{SUBSTR( 'abcdefgh', 0, -2 )}</td>
<td>Returns an error</td>
</tr>
<tr>
<td>\texttt{SUBSTR( 'abcdefg', -2, 2 )}</td>
<td>Returns an empty string</td>
</tr>
</tbody>
</table>

\textbf{SUM function [Aggregate]}

Returns the total of the specified expression for each group of rows.

\textbf{Syntax 1}

\texttt{SUM( [ DISTINCT ] expression )}

\textbf{Parameters}

- \texttt{expression} \hspace{1cm} The name of the expression to be summed. This is commonly a column name.

- \texttt{DISTINCT expression} \hspace{1cm} Computes the sum of the unique values of \texttt{expression} within each group.

\textbf{Returns}

- INTEGER
- DOUBLE
- NUMERIC
Remarks

Rows where the specified expression is NULL are not included.

Returns NULL for a group containing no rows.

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]” on page 308.

See also

- “COUNT function [Aggregate]” on page 317
- “AVG function [Aggregate]” on page 305

Example

The following statement returns the value 3749146.740.

```sql
SELECT SUM( Salary )
FROM GROUPFO.Employees;
```

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” [SQL Anywhere Server - SQL Reference]
- “SYSDATETIMEOFFSET function [Date and time]” [SQL Anywhere Server - SQL Reference]

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.

```sql
SELECT CAST ( '2009-04-03 14:45:12.123-04:00' AS datetimeoffset ) AS EDT,
SWITCHOFFSET( EDT,'-07:00' ) AS PDT;
```
SYNC_PROFILE_OPTION_VALUE function [System]

Returns the value of the option corresponding to the given option name. Not supported by UltraLite Java edition databases.

Syntax

\[
\text{SYNC_PROFILE_OPTION_VALUE}(\text{profile}\_\text{name}, \text{option}\_\text{name})
\]

Parameters

- **profile\_name**  The name of the sync profile to inspect.
- **option\_name**  The name of the option to retrieve the corresponding value for.

Returns

Returns the value of the option corresponding to the given option name.

Remarks

Option names with periods will retrieve values from a sublist with the given base option name before the period, and the given sublist option name after the period.

See also

- “ML_GET_SERVER_NOTIFICATION function [System]” on page 350

Example

Consider the profile:

\[
\text{MobiLinkUid=joe;Stream=tcpip\{host=sybase;port=1234\};Ping=1}
\]

- **MobiLinkUid**  joe
- **Stream**  tcpip\{host=sybase;port=1234\}
- **Stream.host**  sybase
- **Stream.port**  1234
- **Ping**  1

TAN function [Numeric]

Returns the tangent of a number.

Syntax

\[
\text{TAN}( \text{numeric-expression} )
\]
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 316
- “SIN function [Numeric]” on page 371

Example

The following statement returns the value of the tan of 0.52.

```
SELECT TAN( 0.52 );
```

**TODATETIMEOFFSET function [Date and time]**

Converts a TIMESTAMP value to a TIME STAMP WITH TIME ZONE value using the specified time zone offset.

Syntax

```
TODATETIMEOFFSET( timestamp-expression, time-zone-offset )
```

Parameters

- **timestamp-expression** The TIMESTAMP expression to be converted.

- **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.

Returns

TIMESTAMP WITH TIME ZONE

See also

- “TIMESTAMP WITH TIME ZONE data type” [SQL Anywhere Server - SQL Reference]

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.

### Example

The following statements return the current day according to the system clock.

```
SELECT TODAY( * );
SELECT CURRENT DATE;
```

**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
- LONG VARCHAR

### See also

- “LTRIM function [String]” on page 346
- “RTRIM function [String]” on page 366
- “UltraLite string functions” on page 299

### Example

The following statement returns the value chocolate with no leading or trailing blanks.

```
SELECT TRIM( ' chocolate ' );
```
**TRUNCNUM function [Numeric]**

Truncates a number at a specified number of places after the decimal point.

**Syntax**

```
{ 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**

If any parameter is NULL, the result is NULL.

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 366

**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 );
```

---

**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, or LONG VARCHAR corresponding to the data type of the argument.

Remarks

This function is identical to the UCASE function.

See also

- “UPPER function [String]” on page 390
- “LCASE function [String]” on page 340
- “UltraLite string functions” on page 299

Example

The following statement returns the value CHOCOLATE.

```sql
SELECT UCASE( 'ChocoLate' );
```

**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, or LONG VARCHAR corresponding to the data type of the argument.

Remarks

This function is identical to the UCASE function.

See also

- “UCASE function [String]” on page 389
- “LCASE function [String]” on page 340
- “LOWER function [String]” on page 345
- “UltraLite string functions” on page 299

Example

The following statement returns the value CHOCOLATE.

```sql
SELECT UPPER( 'ChocoLate' );
```
UUIDTOSTR function [String]

Converts a unique identifier value (UUID, also known as GUID) to a string value.

**Syntax**

```sql
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 uniqueidentifier, NULL is returned.

This function is useful for viewing a UUID value.

**See also**

- “NEWID function [Miscellaneous]” on page 354
- “STRTOUUID function [String]” on page 382
- “UltraLite string functions” on page 299

**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:

```
SELECT UUIDTOSTR(pk), c1 FROM mytab;
```

The UUIDTOSTR function is not needed for databases created with version 9.0.2 or later.

**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 322.
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 321.

See also
For information about casting data types, see “CAST function [Data type conversion]” on page 308.

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);
```

### 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.

**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 308.

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 322
- “DATEADD function [Date and time]” on page 321

Example

The following statements both return -4.

```sql
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

Example

The following statement returns the value 1998-06-12.

```
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.

```
SELECT YMD( 1999, 15, 1 );
```

UltraLite SQL statements

The SQL statements supported by UltraLite SQL are a subset of the statements supported by SQL Anywhere databases.

Before you begin

- Tables in UltraLite do not support the concept of an owner. As a convenience for existing SQL and for SQL that is programmatically generated, UltraLite still allows the syntax owner.table-name. However, the owner is not checked because table owners are not supported in UltraLite.
UltraLite SQL statement documentation follows the same syntax conventions used by SQL Anywhere statements. Ensure you understand these conventions and how they are used to represent SQL syntax.

Using UltraLite SQL creates a transaction. A transaction consists of all changes (insert, update, and delete statements) since the last rollback or commit statement. These changes can be made permanent by executing a COMMIT. A ROLLBACK statement causes the changes to be removed.

See also

- “SQL statements” [SQL Anywhere Server - SQL Reference]
- “Syntax conventions” [SQL Anywhere Server - SQL Reference]
- “UltraLite transaction processing” on page 454
- “COMMIT statement [UltraLite]” on page 406
- “ROLLBACK statement [UltraLite]” on page 429

UltraLite statement categories

SQL statements are organized and identified by the initial word in a statement, which is almost always a verb. This action-oriented syntax makes the nature of the language into a set of imperative statements (commands) to the database. In UltraLite, supported SQL statements can be classified as follows:

- **Data retrieval statements** Also known as queries. These statements allow select rows of data expressions from tables. Data retrieval is achieved with the SELECT statement.

- **Data manipulation statements** Allow you to change content in the database. Data manipulation is achieved with the following statements:
  - INSERT
  - UPDATE
  - DELETE

- **Data definition statements** Allow you to define the structure or schema of a database. The schema can be changed with the following statements:
  - ALTER DATABASE SCHEMA FROM FILE
  - CREATE INDEX
  - CREATE TABLE
  - DROP INDEX
  - DROP TABLE
  - ALTER TABLE
  - TRUNCATE TABLE
● **Transaction control statements**  Allow you to control transactions within your UltraLite application. Transaction control is achieved with the following statements:

  ○ CHECKPOINT
  ○ COMMIT
  ○ ROLLBACK

● **Synchronization management**  Allow you to temporarily control synchronization with a MobiLink server. Synchronization management is achieved with:

  ○ START SYNCHRONIZATION DELETE
  ○ STOP SYNCHRONIZATION DELETE
  ○ CREATE PUBLICATION
  ○ ALTER PUBLICATION
  ○ DROP PUBLICATION

See also

● “Expressions in UltraLite” on page 256
● “Operators in UltraLite” on page 272
● “SELECT statement [UltraLite]” on page 429
● “INSERT statement [UltraLite]”
● “UPDATE statement [UltraLite]”
● “DELETE statement [UltraLite]”
● “ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]”
● “CREATE INDEX statement [UltraLite]”
● “CREATE TABLE statement [UltraLite]”
● “DROP INDEX statement [UltraLite]”
● “DROP TABLE statement [UltraLite]”
● “ALTER TABLE statement [UltraLite]”
● “TRUNCATE TABLE statement [UltraLite]”
● “CHECKPOINT statement [UltraLite]”
● “COMMIT statement [UltraLite]”
● “ROLLBACK statement [UltraLite]”
● “START SYNCHRONIZATION DELETE statement [UltraLite]”
● “STOP SYNCHRONIZATION DELETE statement [UltraLite]”
● “CREATE PUBLICATION statement [UltraLite]”
● “ALTER PUBLICATION statement [UltraLite]”
● “DROP PUBLICATION statement [UltraLite]”

---

**ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]**

Modifies the schema definition of an existing UltraLite database using a SQL script.

**Note**

This statement is not supported by UltraLite Java edition databases.
Syntax

ALTER DATABASE SCHEMA FROM FILE filename

Parameters

filename  Defines the name and path to the SQL script used to upgrade the schema of an existing UltraLite database.

Remarks

Use either ulinit or ulunload to extract the DDL statements required for your script. By using these utilities, you ensure that the DDL statements are syntactically correct. Use ulinit (-l logfile option) or ulunload (using the -n -s output-file options).

Backup the database before executing this statement.

The character set of the SQL script file must match the character set of the database you want to upgrade.

Ensure that your device is not reset while this statement is executing. If you reset the device during a schema upgrade, the UltraLite database becomes unusable.

Any rows that do not fit into the schema will be dropped (for instance if a uniqueness constraint is added and multiple rows contain the same values, all but one row will be dropped). In this case, the SQL_ROW_DROPPED_DURING_SCHEMA_UPGRADE warning is generated. You can use this warning to detect the error and restore the database from the backup version.

See also

● “UltraLite database schemas” on page 45
● “Deploying UltraLite database schema upgrades” on page 115
● “UltraLite Initialize Database utility (ulinit)” on page 198
● “UltraLite Database Unload utility (ulunload)” on page 216

Example

The following statement modifies the schema of the database using a SQL script, MySchema.sql:

ALTER DATABASE SCHEMA FROM FILE 'MySchema.sql'

ALTER PUBLICATION statement [UltraLite]

Alters a publication.

Syntax

ALTER PUBLICATION publication-name alterpub-clause

alterpub-clause :

ADD TABLE table-name [ WHERE search-condition ]
| ALTER TABLE table-name [ WHERE search-condition ]
| { DROP | DELETE } TABLE table-name
| RENAME publication-name

ALTER DATABASE SCHEMA FROM FILE filename
Remarks

A publication identifies data in a remote database that is to be synchronized.

Side effects

Automatic commit.

See also

- “Search conditions in UltraLite” on page 263
- “UltraLite client synchronization design” on page 68
- “CREATE PUBLICATION statement [UltraLite]” on page 409
- “DROP PUBLICATION statement [UltraLite]” on page 420
- “START SYNCHRONIZATION DELETE statement [UltraLite]” on page 432
- “STOP SYNCHRONIZATION DELETE statement [UltraLite]” on page 433

Example

The following ALTER PUBLICATION statement adds the Customers table to the pub_contact publication.

```
ALTER PUBLICATION pub_contact
  ADD TABLE Customers
```

**ALTER SYNCHRONIZATION PROFILE statement [UltraLite]**

Alters an UltraLite synchronization profile.

**Syntax**

```
ALTER SYNCHRONIZATION PROFILE sync-profile-name
MERGE sync-option [; ... ]
```

**Parameters**

- **sync-profile-name**  The name of the synchronization profile.
- **MERGE clause** Use this clause to change existing, or add new, options to a synchronization profile.
- **sync-option** A string of one or more synchronization option value pairs, separated by semicolons. For example, 'option1=value1;option2=value2'.
- **sync-option-name** The name of the synchronization profile option.
- **sync-option-value** The value for the synchronization profile option.
Remarks

Synchronization profiles define how an UltraLite database synchronizes with the MobiLink server.

You can use the MERGE clause to make changes to an existing synchronization profile. When using this clause, only the synchronization options that are specified in the MERGE clause are changed. To remove a synchronization option from a synchronization profile, the sync-option string should look like 'option1=;' (to set the option to an empty value).

The STREAM synchronization profile option is different from the other options because its value contains a sub-list. For example: 'STREAM=TCPIP{host=192.168.1.1;port=1234}'. In this case 'host=192.168.1.1;port=1234' is the sub-list. To add or remove a sub-list value, use a period between the STREAM sync-option-name and the sub-option-name. For example, MERGE 'stream.port=5678;stream.host=;compression=zlib' results in a synchronization profile of: stream=TCPIP{port=5678;compression=zlib}. Attempting to set the stream to a new value will replace the entire stream value. For example: MERGE 'stream=HTTPS' results in a synchronization profile of: stream=HTTPS{}.

Side effects

None.

See also

● “Synchronization profile options” on page 213
● “DROP SYNCHRONIZATION PROFILE statement [UltraLite]” on page 420
● “SYNCHRONIZE statement [UltraLite]” on page 434

Example

The following is an example of the ALTER SYNCHRONIZATION PROFILE...REPLACE statement:

```
CREATE SYNCHRONIZATION PROFILE myProfile1;
ALTER SYNCHRONIZATION PROFILE myProfile1
    REPLACE 'publications=p1;uploadonly=on';
```

The following is an example of the ALTER SYNCHRONIZATION PROFILE...MERGE statement.

```
CREATE SYNCHRONIZATION PROFILE myProfile2 'publications=p1;
ALTER SYNCHRONIZATION PROFILE myProfile2
    MERGE 'publications=p2;uploadonly=on';
```

The following examples illustrate the changes that occur after executing a sequence of ALTER SYNCHRONIZATION PROFILE commands with different options.

Suppose myProfile1='MobiLinkUID=mary;ScriptVersion=default'.

After executing ALTER SYNCHRONIZATION PROFILE myProfile1 REPLACE 'MobiLinkPwd=sql;ScriptVersion=1', myProfile1 is 'MobiLinkPwd=sql;ScriptVersion=1'.

After executing ALTER SYNCHRONIZATION PROFILE myProfile1 MERGE 'MobiLinkUID=mary;STREAM=tcpip', myProfile1 is 'MobiLinkPwd=sql;ScriptVersion=1;MobiLinkUID=mary;STREAM=tcpip'.

400 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
After executing `ALTER SYNCHRONIZATION PROFILE myProfile1 MERGE 'MobiLinkUID=;STREAM.host=192.168.1.1;STREAM.port=1234;ScriptVersion=; ', myProfile1 is 'MobiLinkPwd=sql;STREAM=tcpip{192.168.1.1;port=1234}'.

After executing `ALTER SYNCHRONIZATION PROFILE myProfile1 MERGE 'MobiLinkPwd=;Ping=yes;STREAM =HTTP', myProfile1 is 'Ping=yes;STREAM=HTTP'.

After executing `ALTER SYNCHRONIZATION PROFILE myProfile1 MERGE 'STREAM=HTTP{host=192.168.1.1}', myProfile1 is 'Ping=yes;STREAM=HTTP{host=192.168.1.1}'.

**ALTER TABLE statement [UltraLite]**

Modifies a table definition.

**Syntax**

```
ALTER TABLE table-name {
  add-clause
  | modify-clause
  | drop-clause
  | rename-clause
}
```

```
add-clause :
  ADD { column-definition | table-constraint }
```

```
modify-clause :
  ALTER column-definition | sync-constraint
```

```
drop-clause :
  DROP { column-name | CONSTRAINT constraint-name }
```

```
rename-clause :
  RENAME {
    new-table-name
    | [ old-column-name TO ] new-column-name
    | CONSTRAINT old-constraint-name TO new-constraint-name
  }
```

```
column-definition :
  column-name data-type
  [ [ NOT ] NULL ]
  [ DEFAULT column-default ]
  [ UNIQUE ]
```

```
column-default :
  GLOBAL AUTOINCREMENT [( number )]
  AUTOINCREMENT
  CURRENT DATE
  CURRENT TIME
  CURRENT TIMESTAMP
  NULL
```
NEWID( )

constant-value

table-constraint :
[ CONSTRAINT constraint-name ]
{ fkey-constraint | unique-key-constraint }
[ WITH MAX HASH SIZE integer ]

fkey-constraint :
[ NOT NULL ] FOREIGN KEY [ role-name ] ( ordered-column-list )
REFERENCES table-name ( column-name, ... )
[ CHECK ON COMMIT ]

unique-key-constraint :
UNIQUE ( ordered-column-list )

ordered-column-list :
( column-name [ ASC | DESC ], ... )

sync-constraint : SYNCHRONIZE {ON | OFF | ALL | DOWNLOAD}

Parameters

add-clause  Adds a new column or table constraint to the table:

Adds a new column or table constraint to the table:

○ ADD column-definition clause  Adds a new column to the table. If the column has a default
value, all rows in the new column are populated with that default value. For descriptions of the
keywords and subclauses for this clause, see “CREATE TABLE statement [UltraLite]” on page 411.

○ ADD table-constraint clause  Adds a constraint to the table. The optional constraint name allows
you to modify or drop individual constraints at a later time, rather than having to modify the entire
table constraint. For descriptions of the keywords and subclauses for this clause, see “CREATE
TABLE statement [UltraLite]” on page 411.

When adding a new unique constraint, all constraint columns must be non nullable. To add a unique
constraint, alter the column to be NOT NULL.

Note
You cannot add a primary key in UltraLite.

modify-clause  Change a single column definition. You cannot use primary keys in the column-
definition when part of an ALTER statement. 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. For
a full explanation of column-definition, see “CREATE TABLE statement [UltraLite]” on page 411.

drop-clause  Delete a column or a table constraint:

○ DROP column-name  Delete the column from the table. If the column is contained in any index,
uniqueness constraint, foreign key, or primary key, then the object must be deleted before UltraLite
can delete the column.
- **DROP CONSTRAINT** *table-constraint*  
  Delete the named constraint from the table definition. For a full explanation of *table-constraint*, see “CREATE TABLE statement [UltraLite]” on page 411.

  **Note**  
  You cannot drop a primary key in UltraLite.

**rename-clause**  
Change the name of a table, column, or constraint:

- **RENAME** *new-table-name*  
  Change the name of the table to *new-table-name*. Any applications using the old table name must be modified. Foreign keys that were automatically assigned the old table name will not change names.

- **RENAME** *old-column-name* TO *new-column-name*  
  Change the name of the column to the *new-column-name*. Any applications using the old column name will need to be modified.

- **RENAME** *old-constraint-name* TO *new-constraint-name*  
  Change the name of the constraint to the *new-constraint-name*. Any applications using the old constraint name need to be modified.

  **Note**  
  You cannot rename a primary key in UltraLite.

**column-constraint**  
A column constraint restricts the values the column can hold to ensure the integrity of data in the database. A column constraint can only be UNIQUE.

- **UNIQUE**  
  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 may have more than one unique constraint.

**sync-constraint clause**  
Specify a sync constraint to determine whether a table can be synchronized or not and whether all rows are uploaded, just changes to the table are uploaded, or no changes to the table are uploaded.

- **SYNCHRONIZE ON**  
  Default setting - the table can be synchronized and only changes to the table are sent in the upload.

- **SYNCHRONIZE OFF**  
  The table cannot be synchronized and it is an error to include the table in a publication.

- **SYNCHRONIZE ALL**  
  The table can be synchronized and all rows in the table are sent in the upload. This constraint is not supported by UltraLite Java edition databases.

- **SYNCHRONIZE DOWNLOAD**  
  The table can be synchronized with changes to the consolidated database but no local changes are uploaded.

**Remarks**  
Only one *table-constraint* or *column-constraint* can be added, modified, or deleted in one ALTER TABLE statement.
The role name is the name of the foreign key. The main function of the role-name is to distinguish two foreign keys to the same table. Alternatively, you can name the foreign key with CONSTRAINT constraint-name. However, do not use both methods to name a foreign key.

You cannot MODIFY a table or column constraint. To change a constraint, you must DELETE the old constraint and ADD the new constraint.

ALTER TABLE cannot execute if a statement that affects the table is already being referenced by another request or query. Similarly, UltraLite does not process requests referencing the table while that table is being altered. Furthermore, you cannot execute ALTER TABLE when the database includes active queries or uncommitted transactions.

For UltraLite.NET users: You cannot execute this statement unless you also call the Dispose method for all data objects (for example, ULDataReader). See “ULBulkCopy.Dispose method [UltraLite.NET]” [UltraLite -.NET Programming].

Statements are not released if schema changes are initiated at the same time.

See also

- “UltraLite database schemas” on page 45
- “CREATE TABLE statement [UltraLite]” on page 411
- “DROP TABLE statement [UltraLite]” on page 421
- “UltraLite SQL data types” on page 275
- “Table alteration” [SQL Anywhere Server - SQL Usage]
- “Table and column constraints” [SQL Anywhere Server - SQL Usage]
- “Partition sizes” on page 67
- “Methods for finding the last assigned GLOBAL AUTOINCREMENT value” on page 66

Examples

The following statement drops the Street column from a fictitious table called MyEmployees.

    ALTER TABLE MyEmployees
    DROP Street

The following example changes the Street column of the fictitious table, MyCustomers, to hold approximately 50 characters.

    ALTER TABLE MyCustomers
    ALTER Street CHAR(50)

**ALTER USER statement [UltraLite]**

Alters user settings.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>This statement is not supported by UltraLite Java edition databases.</td>
</tr>
</tbody>
</table>

**Syntax 1**

    ALTER USER user-name [ IDENTIFIED BY password ]
Parameters

user-name  The name of the user.

IDENTIFIED BY clause  The password for the user.

Remarks

● 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

Side effects

None.

See also

● “CREATE USER statement [UltraLite]” on page 417
● “DROP USER statement [UltraLite]” on page 422

Example

The following alters a user named SQLTester. The password is set to "welcome".

    ALTER USER SQLTester IDENTIFIED BY welcome

CHECKPOINT statement [UltraLite]

Checkpoints the database.

Note

This statement is not supported by UltraLite Java edition databases.

Syntax

    CHECKPOINT

Remarks

You can use the CHECKPOINT statement as a trigger for a commit flush. A commit flush writes uncommitted transactions to storage.
If you are using the embedded SQL API, you can also use the ULCheckpoint method. If you are writing a C++ component application, you can also use the Checkpoint method on a connection object. All other APIs must use this statement.

Side effects
While this statement flushes any pending committed transactions to storage, it does not commit or flush current transactions.

See also
- “Flush single or grouped transactions” on page 455
- “COMMIT statement [UltraLite]” on page 406
- “UltraLite COMMIT_FLUSH connection parameter” on page 163
- “ULCheckpoint method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULConnection.Checkpoint method [UltraLite C++]” [UltraLite - C and C++ Programming]

Example
The following statement performs a checkpoint of the database:

    CHECKPOINT

**COMMIT statement [UltraLite]**

Makes the changes to the database permanent.

Syntax

```
COMMIT [ WORK ]
```

Remarks
Using UltraLite SQL creates a transaction. A transaction consists of all changes (INSERTs, UPDATEs, and DELETEs) since the last ROLLBACK or COMMIT. The COMMIT statement ends the current transaction and makes all changes made during the transaction permanent in the database.

Changes to the database objects using the ALTER, CREATE, and DROP statements are committed automatically.

See also
- “CHECKPOINT statement [UltraLite]” on page 405
- “ROLLBACK statement [UltraLite]” on page 429

Example
The following statement makes the changes in the current transaction permanent in the database:

    COMMIT
**CREATE INDEX statement [UltraLite]**

Creates an index on a specified table.

**Syntax**

```
CREATE [ UNIQUE ] INDEX [ IF NOT EXISTS ] [ index-name ]
ON table-name ( ordered-column-list )
[ WITH MAX HASH SIZE integer ]

ordered-column-list :
( column-name [ ASC | DESC ], ... )
```

**Parameters**

- **UNIQUE**  
  The UNIQUE attribute ensures that there are not two rows in the table with identical values in all the columns in the index. Each index key must be unique or contain a NULL in at least one column.

- **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.

- **ordered-column-list**  
  An ordered list of columns. Column values in the index can be sorted in ascending or descending order.

- **WITH MAX HASH SIZE**  
  Sets the hash size (in bytes) for this index. This value overrides the default MaxHashSize property in effect for the database. For more information about the default size, see “Reading database properties” on page 35.

**Remarks**

UltraLite automatically creates indexes for primary keys and for unique constraints.

Indexes can improve query performance by providing quick ways for UltraLite to look up specific rows. Conversely, because they have to be maintained, indexes may slow down synchronization and INSERT, DELETE, and UPDATE statements.

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 remove it with DROP INDEX.

Indexes use space in the database. Also, the additional work required to maintain indexes can affect the performance of data modification operations. For these reasons, you should avoid creating indexes that do not improve query performance.
UltraLite does not process requests or queries referencing the index while the CREATE INDEX statement is being processed. Furthermore, you cannot execute CREATE INDEX when the database includes active queries or uncommitted transactions.

UltraLite can provide developers with execution plans to optimize queries. See “Execution plans in UltraLite” on page 446.

For UltraLite.NET users: You cannot execute this statement unless you also call the Dispose method for all data objects (for example, ULDataReader). See “ULBulkCopy.Dispose method [UltraLite.NET]” [UltraLite - .NET Programming].

Statements are not released if schema changes are initiated at the same time. See “UltraLite database schemas” on page 45.

Side effects

- Automatic commit.

See also

- “UltraLite performance tips” on page 439
- “DROP INDEX statement [UltraLite]” on page 419
- “UltraLite max_hash_size creation parameter” on page 141
- “UltraLite indexes” on page 52

Example

The following statement creates a two-column index on the Employees table.

```sql
CREATE INDEX employee_name_index
ON Employees ( Surname, GivenName )
```

The following statement creates an index on the SalesOrderItems table for the ProductID column.

```sql
CREATE INDEX item_prod
ON SalesOrderItems ( ProductID )
```

The following scenario illustrates the effects of MAX HASH SIZE on an UltraLite Java edition database, given an Employees table that contains an Initials column that is VARCHAR( 3 ) and an EmployeeID column that is TINY.

The following statement completely hashes all values when only ASCII7 characters are used:

```sql
CREATE INDEX ascii_a ON Employees( Initials ) WITH MAX HASH SIZE 3
```

The following statement completely hashes all values no matter what characters they contain:

```sql
CREATE INDEX unicode_a ON Employees( Initials ) WITH MAX HASH SIZE 9
```

The following statement only hashes the Initials values even when only ASCII characters are used because the first 9 bytes for Initials are reserved:

```sql
CREATE INDEX compound_1 ON Employees( Initials, EmployeeID ) WITH MAX HASH SIZE 9
```
The following statement completely hashes both Initials and EmployeeID values:

```
CREATE INDEX compound_2 ON Employees( Initials, EmployeeID ) WITH MAX HASH SIZE 10
```

**CREATE PUBLICATION statement [UltraLite]**

Creates a publication.

**Syntax**

```
CREATE PUBLICATION [ IF NOT EXISTS ]publication-name
( TABLE table-name [ WHERE search-condition ], ... )
```

**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.

- **TABLE clause**  Use the table to include a TABLE in the publication. There is no limit to the number of TABLE clauses.

- **WHERE clause**  If a WHERE clause is specified, only rows satisfying `search-condition` are considered for upload from the associated table during synchronization.

If you do not specify a WHERE clause, every row in the table that has changed in UltraLite since the last synchronization is considered for upload.

**Remarks**

A publication identifies synchronized data in an UltraLite remote database.

A publication establishes tables that are synchronized during a single synchronization operation, and determines which data is uploaded to the MobiLink server. The MobiLink server may send back rows for these (and only these) tables during its download session; however, rows that are downloaded do not have to satisfy the WHERE clause for a table.

Only entire tables can be published. You cannot publish specific columns of a table in UltraLite.

**Side effects**

- Automatic commit.

**See also**

- “Search conditions in UltraLite” on page 263
- “UltraLite clients” on page 63
- “DROP PUBLICATION statement [UltraLite]” on page 420
- “ALTER PUBLICATION statement [UltraLite]” on page 398
- “Search conditions in UltraLite” on page 263

**Example**

The following statement publishes all the columns and rows of two tables.
CREATE PUBLICATION pub_contact (TABLE Contacts, TABLE Customers)

The following statement publishes only the rows of the Customers table where the State column contains MN.

CREATE PUBLICATION pub_customer (TABLE Customers WHERE State = 'MN')

CREATE SYNCHRONIZATION PROFILE statement

[UltraLite]

Creates or replaces an UltraLite synchronization profile.

Syntax

CREATE [OR REPLACE] SYNCHRONIZATION PROFILE sync-profile-name sync-option [;...]

sync-option :
sync-option-name = sync-option-value

sync-option-name : string

sync-option-value : string

Parameters

- **OR REPLACE clause** If the named synchronization profile already exists, then it will be replaced. If the profile does not exist, it will be created.

- **sync-profile-name** The name of the synchronization profile.

- **sync-option** A string of one or more synchronization option value pairs, separated by semicolons. For example, 'option1=value1;option2=value2'.

- **sync-option-name** The name of the synchronization profile option.

- **sync-option-value** The value for the synchronization profile option.

Remarks

Synchronization profiles define how an UltraLite database synchronizes with the MobiLink server.

You can use the REPLACE clause to make changes to an existing synchronization profile. This clause replaces the contents of the synchronization profile with whatever is contained in the new sync-option string. This approach is the same as dropping the synchronization profile and then creating one with the same name but using the new string. Therefore, a synchronization profile does not need to contain a full synchronization definition because parameters can be merged in or overridden at synchronization time.
The STREAM synchronization profile option is different from the other options because its value contains a sub-list. For example: 'STREAM=TCPIP{host=192.168.1.1;port=1234}'. In this case 'host=192.168.1.1;port=1234' is the sub-list. To add or remove a sub-list value, use a period between the STREAM sync-option-name and the sub-option-name. For example, MERGE 'stream.port=5678;stream.host=;compression=zlib' results in a synchronization profile of: stream=TCPIP(port=5678;compression=zlib). Attempting to set the stream to a new value will replace the entire stream value. For example: MERGE 'stream=HTTPS' results in a synchronization profile of: stream=HTTPS{}.

Side effects
None.

See also
- “Synchronization profile options” on page 213
- “ALTER SYNCHRONIZATION PROFILE statement [UltraLite]” on page 399
- “DROP SYNCHRONIZATION PROFILE statement [UltraLite]” on page 420
- “SYNCHRONIZE statement [UltraLite]” on page 434

Example
The following creates a synchronization profile called Test1.

```
CREATE SYNCHRONIZATION PROFILE Test1 
'MobiLinkUid=mary;Stream=TCPIP{host=192.168.1.1;port=1234}'
```

CREATE TABLE statement [UltraLite]

Creates a table.

Syntax
```
CREATE TABLE [ IF NOT EXISTS ] table-name ( 
   { column-definition | table-constraint| sync-constraint }, ... 

column-definition :
column-name  data-type
[ [ NOT | NULL ]
[ DEFAULT column-default]
[STORE AS FILE (file-name-column) [CASCADE DELETE ]
[ column-constraint ]

column-default :
AUTOFILENAME(prefix,extension)
GLOBAL AUTOINCREMENT [( number ) ]
AUTOINCREMENT
CURRENT DATE
CURRENT TIME
CURRENT TIMESTAMP
CURRENT UTC TIMESTAMP
NULL
```
NEWID()
  constant-value

file-name
  "filename"

column-constraint :
  PRIMARY KEY
  UNIQUE

table-constraint :
  [ CONSTRAINT constraint-name ]
  pkey-constraint
  fkey-constraint
  unique-key-constraint
  [ WITH MAX HASH SIZE integer ]

pkey-constraint :
  PRIMARY KEY [ ordered-column-list ]

fkey-constraint :
  [ NOT NULL ] FOREIGN KEY [ role-name ] ( ordered-column-list )
  REFERENCES table-name ( column-name, ... )
  [ CHECK ON COMMIT ]

unique-key-constraint :
  UNIQUE ( ordered-column-list )

ordered-column-list :
  ( column-name [ ASC | DESC ], ... )

sync-constraint : SYNCHRONIZE { ON | OFF | ALL | DOWNLOAD }

Parameters

IF NOT EXISTS clause  Use this clause to create a table. No changes are made if the named table already exists, and an error is not returned.

column-definition  Defines a column in a table. Available parameters for this clause include:

  ○ column-name  The column name is an identifier. Two columns in the same table cannot have the same name. See “Identifiers in UltraLite” on page 251.

  UltraLite Java edition databases support the partitioning of database files such that external files may now be used to store large BLOB values, with the files referenced by two columns: the first stores the file name and should have a data type of CHAR(size) ... AUTOFilename(...), and the second is used to access the file contents and is declared as a LONG BINARY STORE AS ... data type. The file contents column is read-only.

  ○ data-type  The data type of the column. See “UltraLite SQL data types” on page 275.

  ○ [ NOT ] NULL  If NOT NULL is specified, or if the column is in a PRIMARY KEY or UNIQUE constraint, the column cannot contain NULL in any row. Otherwise, NULL is allowed.
- column-default  Sets the default value for the column. 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 is specified, it is equivalent to DEFAULT NULL. Default options include:

  - AUTOFILENAME  This clause supports the storing of external BLOB files in a partitioned UltraLite Java edition database.

  When partitioning the database, the column designated to store the file names requires the AUTOFILENAME(prefix,extension) clause. This clause specifies how new filenames are to be generated for downloaded BLOB values. The prefix and extension values are string literal constants. See “Connection.OPTION_BLOB_FILE_BASE_DIR variable [BlackBerry] [UltraLiteJ]” [UltraLite - Java Programming].

  - 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 used as a starting point for subsequent inserts.

  Tip
  In UltraLite, the autoincrement value is not set to 0 when the table is created, and AUTOINCREMENT generates negative numbers when a signed data type is used for the column. Therefore, declare AUTOINCREMENT columns as unsigned integers to prevent negative values from being used.

  - GLOBAL AUTOINCREMENT  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. UltraLite supplies default values in a database only from the partition uniquely identified by that database's number.

  Tip
  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.

  See “GLOBAL AUTOINCREMENT columns in UltraLite” on page 64, and “UltraLite global_database_id option” on page 184.

  - [ NOT ] NULL  Controls whether the column can contain NULLs.

  - NEWID( )  A function that generates a unique identifier value. See “NEWID function [Miscellaneous]” on page 354.

  - 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. The fraction of a second is stored to 3 decimal places. The accuracy is limited by the accuracy of the system clock. See “CURRENT TIMESTAMP special value” on page 255.
| CURRENT UTC TIMESTAMP | A TIMESTAMP WITH TIMEZONE value containing the Coordinated Universal Time (UTC) comprised of the year, month, day, hour, minute, second, fraction of a second, and time zone. The fraction of a second is stored to 3 decimal places. The accuracy is limited by the accuracy of the system clock. See “CURRENT UTC TIMESTAMP special value” on page 255.

| CURRENT DATE | Stores the current year, month, and day. See “CURRENT DATE special value” on page 253.

| CURRENT TIME | Stores the current hour, minute, second and fraction of a second. See “CURRENT TIME special value” on page 254.

| constant-value | A constant for the data type of the column. Typically the constant is a number or a string.

- **STORE AS FILE (file-name-column) [CASCADE DELETE]** Supported by UltraLite Java edition databases only.

Specify that a LONG BINARY column is to be stored externally (partitioning the database) and specify the file-name-column to name the column that will be used to store the file names of the externally stored BLOB values. A column with this clause must be of type LONG BINARY and behave as a read-only column.

- **column-constraint clause** Specify a column constraint to restrict the values allowed in a column. A column constraint can be one of:

  - **PRIMARY KEY** When set as part of a column-constraint, the PRIMARY KEY clause sets the column as the primary key for the table. Primary keys uniquely identify each row in a table. By default, columns included in primary keys do not allow NULL.

  - **UNIQUE** 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 may have more than one unique constraint. NULL values are not allowed.

- **table-constraint clause** Specify a table constraint to restrict the values that one or more columns in the table can hold. Use the CONSTRAINT clause to specify an identifier for the table constraint. Table constraints can be in the form of a primary key constraint, a foreign key constraint, or a unique constraint, as defined below:

  - **pkey-constraint clause** Sets the specified column(s) as the primary key for the table. Primary keys uniquely identify each row in a table. Columns included in primary keys cannot allow NULLs.

  - **fkey-constraint clause** Specify a foreign key constraint to restrict values for one or more columns that must match the values in a primary key (or a unique constraint) of another table.

  - **NOT NULL clause** Specify NOT NULL to 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. If at least one value in a multi-column foreign key is NULL, there is no restriction on the values that can be held in other columns of the key.
- **role-name clause** Specify a role-name to name the foreign key. role-name is used to distinguish foreign keys within the same table. Alternatively, you can name the foreign key using CONSTRAINT constraint-name. However, do not use both methods to name a foreign key.

- **REFERENCES clause** Specify the REFERENCES clause to define one or more columns in the primary table to use as the foreign key constraint. Any column-name you specify in a REFERENCES column constraint must be a column in the primary table, and must be subject to a unique constraint or primary key constraint.

- **CHECK ON COMMIT** not supported by UltraLite Java edition databases. Specify CHECK ON COMMIT to cause the database server to wait for a COMMIT before enforcing foreign key constraints. By default, foreign key constraints are enforced immediately during insert, update, or delete operations. However, when CHECK ON COMMIT is set, database changes can be made in any order, even if they violate foreign key constraints, if inconsistent data is resolved before the next COMMIT.

  ○ **unique-key-constraint clause** Specify a unique constraint to identify 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 may have more than one unique constraint.

  ○ **WITH MAX HASH SIZE** Sets the hash size (in bytes) for this index. This value overrides the default MaxHashSize property in effect for the database. To learn the default size, see “Reading database properties” on page 35. Also see “UltraLite max_hash_size creation parameter” on page 141.

- **sync-constraint clause** Specify a sync constraint to determine whether a table can be synchronized or not and whether all rows are uploaded, just changes to the table are uploaded, or no changes to the table are uploaded.

  ○ **SYNCHRONIZE ON** Default setting - the table can be synchronized and only changes to the table are sent in the upload.

  ○ **SYNCHRONIZE OFF** The table cannot be synchronized and it is an error to include the table in a publication.

  ○ **SYNCHRONIZE ALL** The table can be synchronized and all rows in the table are sent in the upload. This constraint is not supported by UltraLite Java edition databases.

  ○ **SYNCHRONIZE DOWNLOAD** The table can be synchronized with changes to the consolidated database but no local changes are uploaded.

**Remarks**

Column constraints are normally used unless the constraint references more than one column in the table. In these cases, a table constraint must be used. If a statement causes 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.

Each row in the table has a unique primary key value.

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.

**Schema changes** Statements are not released if schema changes are initiated at the same time. See “UltraLite database schemas” on page 45.

UltraLite does not process requests or queries referencing the table while the CREATE TABLE statement is being processed. Furthermore, you cannot execute CREATE TABLE when the database includes active queries or uncommitted transactions.

For UltraLite.NET users: You cannot execute this statement unless you also call the Dispose method for all data objects (for example, ULDataReader). See “ULBulkCopy.Dispose method [UltraLite.NET]” [UltraLite - .NET Programming].

**Synchronization of external BLOB columns (UltraLiteJ only)** On the consolidated database, the filename column is stored as a regular CHAR column and the BLOB file column is stored as a regular BLOB (LONG BINARY) column. On a download, the filename column is ignored and a new filename is generated using the database option (Connection.OPTION_BLOB_FILE_BASE_DIR) and the prefix and extension strings specified on the DEFAULT AUTOFILENAME clause. For J2SE value stored in the filename column will match the syntax `<database_option_blob_file_base_dir><prefix><auto generated integer value>.<extension>` and for the BlackBerry the value stored will match the syntax `<prefix><auto generated integer value>.<extension>` Therefore, for the BlackBerry generated filenames are always relative.

**Accessing external BLOB columns (UltraLiteJ only)** Files containing external BLOB values are only opened when the client application attempts to read the column value. At that time, the filename stored in the column designated by the STORE AS FILE clause must be a valid filename. For the BlackBerry, relative filenames are resolved against the database option OPTION_BLOB_FILE_BASE_DIR. If UltraLite determines that a filename does not begin with the prefix "file://", it will prepend the filename with the value of the OPTION_BLOB_FILE_BASE_DIR option before attempting to open it.

A BLOB file is inserted into the database by specifying a filename for the file-name-column column. The inserted filename must be a valid filename and, for the Blackberry, it must follow the format for a fully qualified, absolute path file name as described in the file URL format in IETF RFCs 1738 & 2396 (see the Package javax.microedition.io.file Description of BlackBerry JDE API documentation). On insert, a value for the STORE AS FILE column may not be specified.

Once a file is inserted into the database, the database assumes full control over the file and assumes that no outside modification will be made.

**Side effects**

Automatic commit.
See also

- “Expressions in UltraLite” on page 256
- “DROP TABLE statement [UltraLite]” on page 421
- “CREATE TABLE statement” [SQL Anywhere Server - SQL Reference]
- “UltraLite SQL data types” on page 275
- “Partition sizes” on page 67

Example

The following statement creates a table for a library database to hold book information.

```sql
CREATE TABLE library_books (
  isbn CHAR(20) PRIMARY KEY,
  copyright_date DATE,
  title CHAR(100),
  author CHAR(50),
  location CHAR(50),
  FOREIGN KEY location REFERENCES room
)
```

The following statement creates a table for a library database to hold information about borrowed books. The default value for date_borrowed indicates that the book is borrowed on the day the entry is made. The date_returned column is NULL until the book is returned.

```sql
CREATE TABLE borrowed_book (  
  loaner_name CHAR(100) PRIMARY KEY,
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
  date_returned DATE,
  book CHAR(20),
  FOREIGN KEY (book) REFERENCES library_books (isbn)
)
```

The following statement creates tables for a sales database to hold order and order item information.

```sql
CREATE TABLE Orders (  
  order_num INTEGER NOT NULL PRIMARY KEY,
  date_ordered DATE,
  name CHAR(80)
);
CREATE TABLE Order_item (  
  order_num INTEGER NOT NULL,
  item_num SMALLINT NOT NULL,
  PRIMARY KEY (order_num, item_num),
  FOREIGN KEY (order_num) REFERENCES Orders (order_num)
)
```

CREATE USER statement [UltraLite]

Creates a database user or group.

Note

This statement is not supported by UltraLite Java edition databases.
Syntax

    CREATE USER user-name IDENTIFIED BY password

Parameters

user-name    The name of the user you are creating.

password     The password for the user you are creating.

Remarks

- 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

Side effects

None.

See also

- “ALTER USER statement [UltraLite]” on page 404
- “DROP USER statement [UltraLite]” on page 422

Example

The following example creates a user named SQLTester with the password welcome.

    CREATE USER SQLTester IDENTIFIED BY welcome

DELETE statement [UltraLite]

Deletes rows from a table in the database.

Syntax

    DELETE [ FROM ] table-name[[AS] correlation-name]
        [ WHERE search-condition ]

Parameters

correlation-name  An identifier to use when referencing the table from elsewhere in the statement.

WHERE clause      If a WHERE clause is specified, only rows satisfying search-condition are deleted.
The WHERE clause does not support non-deterministic functions (like RAND) or variables. Nor does this clause restrict columns; columns may need to reference another table when used in a subquery.

Remarks
The way in which UltraLite traces row states is unique. Be sure you understand the implication of deletes and row states.

See also
- “Search conditions in UltraLite” on page 263
- “UltraLite database row state management” on page 453
- “START SYNCHRONIZATION DELETE statement [UltraLite]” on page 432
- “STOP SYNCHRONIZATION DELETE statement [UltraLite]” on page 433

Example
The following statement removes employee 105 from the Employees table.

```
DELETE
FROM Employees
WHERE EmployeeID = 105
```

The following statement removes all data before the year 2000 from the FinancialData table.

```
DELETE
FROM FinancialData
WHERE Year < 2000
```

DROP INDEX statement [UltraLite]

Deletes an index.

Syntax
```
DROP INDEX [ IF EXISTS ] [ table-name.]index-name
```

Remarks
You cannot drop the primary index of a table.

UltraLite does not process requests or queries referencing the index while the DROP INDEX statement is being processed. Furthermore, you cannot execute DROP INDEX when the database includes active queries or uncommitted transactions.

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.

For UltraLite.NET users: You cannot execute this statement unless you also call the Dispose method for all data objects (for example, ULDataReader). See “ULBulkCopy.Dispose method [UltraLite.NET]” [UltraLite - .NET Programming].
Statements are not released if schema changes are initiated at the same time. See “UltraLite database schemas” on page 45.

See also
- “CREATE INDEX statement [UltraLite]” on page 407
- “UltraLite indexes” on page 52

Example
The following statement deletes a fictitious index, fin_codes_idx, on the FinancialData table:

```
DROP INDEX FinancialData.fin_codes_idx
```

**DROP PUBLICATION statement [UltraLite]**

Deletes publications.

**Syntax**
```
DROP PUBLICATION [ IF EXISTS ] publication-name, ...
```

**Remarks**
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.

See also
- “UltraLite client synchronization design” on page 68
- “ALTER PUBLICATION statement [UltraLite]” on page 398
- “CREATE PUBLICATION statement [UltraLite]” on page 409

Example
The following statement drops the pub_contact publication.

```
DROP PUBLICATION pub_contact
```

**DROP SYNCHRONIZATION PROFILE statement [UltraLite]**

Deletes a synchronization profile.

**Syntax**
```
DROP SYNCHRONIZATION PROFILE [ IF EXISTS ] sync-profile-name
```

**Parameters**
- **sync-profile-name** The name of the synchronization profile.
Remarks
Synchronization profiles define how an UltraLite or UltraLite Java edition database synchronizes with the MobiLink server.

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.

Side effects
None.

See also
● “CREATE SYNCHRONIZATION PROFILE statement [UltraLite]” on page 410
● “ALTER SYNCHRONIZATION PROFILE statement [UltraLite]” on page 399
● “SYNCHRONIZE statement [UltraLite]” on page 434

Example
The following example shows the syntax for dropping a synchronization profile called Test1.

    DROP SYNCHRONIZATION PROFILE Test1

DROP TABLE statement [UltraLite]
Removes a table, and all its data, from a database.

Syntax
    DROP TABLE [ IF EXISTS ] table-name

Remarks
The DROP TABLE statement drops the specified table from the database. All data in the table and any indexes and keys are also removed.

UltraLite does not process requests or queries referencing the table, or its indexes, while the DROP TABLE statement is being processed. Furthermore, you cannot execute DROP TABLE when there are active queries or uncommitted transactions.

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.

For UltraLite.NET, you cannot execute this statement unless you also call the Dispose method for all data objects (for example, ULDataReader). See “ULBulkCopy.Dispose method [UltraLite.NET]” [UltraLite .NET Programming].

Statements are not released if schema changes are initiated at the same time. See “UltraLite database schemas” on page 45.
See also
- “ALTER TABLE statement [UltraLite]” on page 401
- “CREATE TABLE statement [UltraLite]” on page 411

Example
The following statement deletes a fictitious table, EmployeeBenefits, from the database:

```
DROP TABLE EmployeeBenefits
```

**DROP USER statement [UltraLite]**

Drops a user.

**Note**
This statement is not supported by UltraLite Java edition databases.

**Syntax**

```
DROP USER userid IDENTIFIED BY password
```

**Parameters**

- **userid**  The user ID of the user you are dropping.
- **password**  The password for the user.

**Remarks**

None.

**Side effects**

None.

See also

- “ALTER USER statement [UltraLite]” on page 404
- “CREATE USER statement [UltraLite]” on page 417

Example
The following example drops the user SQLTester from a database.

```
DROP USER SQLTester
```

**FROM clause [UltraLite]**

Use this clause to specify the tables or views involved in a SELECT statement.

**Note**
This statement is not supported by UltraLite Java edition databases.
Syntax

```
FROM  table-expression, ...
```

```
table-expression :
  table-name [ [ AS ] correlation-name ]
  | ( select-list ) [ AS ] derived-table-name ( column-name, ... )
  | ( table-expression )
  | table-expression join-operator table-expression [ ON search-condition ] ...

join-operator :
  | INNER JOIN
  | CROSS JOIN
  | LEFT OUTER JOIN
  | JOIN
```

Parameters

**table-name**  A base table or temporary table. Tables cannot be owned by different users in UltraLite. If you qualify tables with user ID, the ID is ignored.

**correlation-name**  An identifier to use when referencing the table from elsewhere in the statement. For example, in the following statement, a is defined as the correlation name for the Contacts table, and b is the correlation name for the Customers table.

```
SELECT *
FROM Contacts a, Customers b
WHERE a.CustomerID=b.ID
```

**derived-table-name**  A derived table is a nested SELECT statement in the FROM clause.

Items from the SELECT list of the derived table are referenced by the (optional) derived table name followed by a period (.) and the column name. You can use the column name by itself if it is unambiguous.

You cannot reference derived tables from within the SELECT statement.

**join-operator**  Specify the type of join. If you specify a comma (,), or CROSS JOIN, you cannot specify an ON subclause. If you specify JOIN, you must specify an ON subclause. For INNER JOIN and LEFT OUTER JOIN, the ON clause is optional.

Remarks

When there is no FROM clause, the expressions in the SELECT statement must be a constant expression.

**Note**  Although this description refers to tables, it also applies to 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 specified tables are in the result set, and the number of combinations is usually reduced by JOIN conditions and/or WHERE conditions.

If you do not specify the type of join, and instead list the tables as a comma-separated list, a CROSS JOIN is used, by default.
For INNER joins, restricting results of the join using an ON clause or WHERE clause returns equivalent results. For OUTER joins, they are not equivalent.

**Note**
UltraLite does not support KEY JOINS nor NATURAL joins.

**See also**
- “Subqueries in expressions” on page 261
- “Joins: Retrieving data from several tables” [SQL Anywhere Server - SQL Usage]
- “DELETE statement [UltraLite]” on page 418
- “SELECT statement [UltraLite]” on page 429
- “UPDATE statement [UltraLite]” on page 437

**Example**

The following are valid FROM clauses:

```sql
... FROM Employees ... 
...
FROM Customers 
CROSS JOIN SalesOrders 
CROSS JOIN SalesOrderItems 
CROSS JOIN Products 
...
```

The following query uses a derived table to return the names of the customers in the Customers table who have more than three orders in the SalesOrders table:

```sql
SELECT Surname, GivenName, number_of_orders
FROM Customers JOIN 
    ( SELECT CustomerID, COUNT(*)
    FROM SalesOrders 
    GROUP BY CustomerID )
AS sales_order_counts( CustomerID, number_of_orders )
ON ( Customers.id = sales_order_counts.CustomerID )
WHERE number_of_orders > 3
```

**INSERT statement [UltraLite]**

Inserts rows into a table.

**Syntax**

```sql
INSERT [ INTO ]
    table-name [ ( column-name, ... ) ]
{ VALUES ( expression, ... ) | select-statement }
```

**Remarks**

The INSERT statement can be used to insert a single row, or to insert multiple rows from a query result set.
If columns are specified, values are inserted one for one into the specified columns. If the list of column names is not specified, values are inserted into the table columns in the order in which they appear in the table (the same order as retrieved with SELECT *). Rows are inserted into the table in an arbitrary order.

Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive.

See also

- “SELECT statement [UltraLite]” on page 429

Example

The following statement adds an Eastern Sales department to the database.

```sql
INSERT
INTO Departments ( DepartmentID, DepartmentName )
VALUES ( 230, 'Eastern Sales' )
```

The following statement adds the values of a and b into mytable from othertable where the value of c in othertable is greater than 10.

```sql
INSERT INTO mytable( col1, col2 ) SELECT a, b FROM othertable WHERE c > 10
```

### LOAD TABLE statement [UltraLite]

Imports bulk data into a database table from an external file.

**Syntax**

```sql
LOAD [ INTO ] TABLE [ owner.]table-name
( column-name, ... )
FROM stringfilename
[ load-option ... ]
```

*load-option* :

- CHECK CONSTRAINTS { ON | OFF }
- COMPUTES { ON | OFF }
- DEFAULTS { ON | OFF }
- DELIMITED BY string
- ENCODING encoding
- ESCAPES { ON }
- FORMAT { ASCII | TEXT }
- ORDER { ON | OFF }
- QUOTES { ON | OFF }
- SKIP integer
- STRIP { ON | OFF | BOTH }
- WITH CHECKPOINT { ON | OFF }

*comment-prefix* : string

*encoding* : string
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.

- **FROM string-filename**  Use this to specify a file from which to load the data. The string-filename 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 statement to load data from the file c:\temp\input.dat into the Employees table is:

    ```
    LOAD TABLE Employees
    FROM 'c:\temp\input.dat' ...
    ```

  - 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.

- **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:
  
  - **CHECK CONSTRAINTS clause**  This clause controls whether constraints are checked during loading. CHECK CONSTRAINTS is ON by default, but the Unload utility (ulunload) 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.

  - **COMPUTES clause**  This option is processed but ignored by UltraLite.

  - **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 formatting requirements of other SQL strings apply. To specify tab-delimited values, you could specify the hexadecimal escape sequence for the tab character (9), ... DELIMITED BY '\x09' ....

  - **ENCODING clause**  This clause specifies the character encoding used for the data being loaded into the database. UltraLite does not perform character set translation: the encoding of the data file must match the database.
○ **ESCAPES clause**  ESCAPES is always ON, therefore characters following the backslash character 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.

○ **FORMAT clause**  This clause specifies the format of the data source you are loading data from. With TEXT, 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. ASCII is also supported.

○ **QUOTES clause**  This clause specifies whether strings are enclosed in quotes. UltraLite only supports ON, therefore the LOAD TABLE statement expects strings to be enclosed in quote characters. The quote character is an apostrophe (single quote). 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.

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, 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',''''

○ **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.

○ **STRIP clause**  This clause is processed but ignored. This clause specifies whether unquoted values should have leading or trailing blanks stripped off before they are inserted. The STRIP option accepts the following options:

  ● **STRIP ON**  Strip leading blanks.

  ● **STRIP OFF**  Do not strip off leading or trailing blanks.

  ● **STRIP BOTH**  Strip both leading and trailing blanks.

○ **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 the statement.

**Remarks**

This statement also provides support for handling the output of the SQL Anywhere dbunload utility (the reload.sql file). LOAD TABLE is only available using DBISQL on Windows.
Note
This statement is not supported by UltraLite Java edition databases.

The recommended method for unloading and reloading UltraLite databases is to use the ulunload and ulload utilities. Note also that the ulinit utility can load schema and data directly from a SQL Anywhere database.

LOAD TABLE allows efficient mass insertion into a database table from a file. It is provided primarily as a means of supporting the output of the SQL Anywhere dbunload utility (the reload.sql file).

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 only specifies 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 value, and NULLs are allowed, a NULL is used.

● if column c has no default value and does not allow NULLs, either a zero (0) or an empty string (''), is used, or an error is returned, depending on the data type of the column.

Side effects
Automatic commit.

See also
● “INSERT statement [UltraLite]” on page 424
● “UltraLite Database Unload utility (ulunload)” on page 216
● “Unload utility (dbunload)” [SQL Anywhere Server - Database Administration]

Example
Following is an example of LOAD TABLE. First, you create a table, and then load data into it using a file called input.txt.

```
CREATE TABLE t( a CHAR(100) primary key, let_me_default INT DEFAULT 1, c CHAR(100) )
```

Following is the content of a file called input.txt:
The following LOAD statement loads the file called *input.txt*:

```
LOAD TABLE T ( c, a ) FROM 'input.txt' FORMAT TEXT DEFAULTS ON
```

The command `SELECT * FROM t` 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>

**ROLLBACK statement [UltraLite]**

Ends a transaction and reverts any changes made to data since the last COMMIT or ROLLBACK statement was executed.

**Syntax**

```
ROLLBACK [ WORK ]
```

**Remarks**

Using UltraLite SQL creates a transaction. A transaction consists of all changes (INSERTs, UPDATEs, and DELETEs) since the last ROLLBACK or COMMIT. The ROLLBACK statement ends the current transaction and undoes all changes made to the database since the previous COMMIT or ROLLBACK.

**See also**

- “COMMIT statement [UltraLite]” on page 406

**Example**

The following statement rolls the database back to the state it was in at the previous commit:

```
ROLLBACK
```

**SELECT statement [UltraLite]**

Retrieves information from the database.

**Syntax**

```
SELECT [ DISTINCT ] [ row-limitation ]
  select-list
  [ FROM table-expression, ... ]
  [ WHERE search-condition ]
  [ GROUP BY group-by-expression, ... ]
  [ ORDER BY order-by-expression, ... ]
  [ FOR { UPDATE | READ ONLY } ]
  [ OPTION ( FORCE ORDER ) ]
```
row-limitation:
- FIRST
- TOP \( n \) [START AT \( m \)]

select-list:
- expression \([ \ AS \] alias-name \], ...

order-by-expression:
- \{ integer | expression \} [ASC | DESC]

Parameters

DISTINCT clause Specify DISTINCT to eliminate duplicate rows from the results. If you do not specify DISTINCT, all rows that satisfy the clauses of the SELECT statement are returned, including duplicate rows. Many statements take significantly longer to execute when DISTINCT is specified, so you should reserve DISTINCT for cases where it is necessary.

row-limitation clause Use row limitations to return a subset of the results. For example, specify FIRST to retrieve the first row of a result set. Use TOP \( n \) to return the first \( n \) rows of the results. Specify START AT \( m \) to control the location of the starting row when retrieving the TOP \( n \) rows. To order the rows so that these clauses return meaningful results, specify an ORDER BY clause for the SELECT statement.

select-list A list of expressions specifying what to retrieve from the database. Usually, the expressions in a SELECT list are column names. However, they can be other types of expressions, such as functions. Use an asterisk (*) to select all columns of all tables listed in the FROM clause. Optionally, you can define an alias for each expression in the select-list. Using an alias allows you to reference the select-list expressions from elsewhere in the query, such as from within the WHERE and ORDER BY clauses.

FROM clause Rows are retrieved from the tables and views specified in the table-expression. See “FROM clause [UltraLite]” on page 422.

WHERE clause If a WHERE clause is specified, only rows satisfying search-condition are selected. See “Search conditions in UltraLite” on page 263.

GROUP BY clause The result of the query that has a GROUP BY clause contains one row for each distinct set of values in the GROUP BY expression. The resulting rows are often referred to as groups since there is one row in the result for each group of rows from the table list. Aggregate functions can be applied to the rows in these groups. NULL is considered to be a unique value if it occurs.

ORDER BY clause This clause sorts the results of a query according to the expression specified in the clause. Each expression in the ORDER BY clause can be sorted in ascending (ASC) or descending (DESC) order (the default). If the expression is an integer \( n \), then the query results are sorted by the \( n \)th expression 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, UltraLite returns rows in whatever order is most efficient.

UltraLite does not support the ordering of LONG VARCHAR or LONG BINARY values.

FOR clause This clause has two variations that control the query's behavior:
FOR READ ONLY This clause indicates the query is not being used for updates. You should specify this clause whenever possible, since UltraLite can sometimes achieve better performance when it is known that a query is not going to be used for updates. For example, UltraLite could perform a direct table scan when it learns that read-only access is required. FOR READ ONLY is the default behavior. See “Direct page scans” on page 448.

FOR UPDATE This clause allows the query to be used for updates. This clause must be explicitly specified otherwise updates are not permitted (FOR READ ONLY is the default behavior).

OPTION (FORCE ORDER) clause This clause is not recommended for general use. It overrides the UltraLite choice of the order in which to access tables, and requires that UltraLite access the tables in the order they appear in the query. Only use this clause when the query order is determined to be more efficient than the UltraLite order.

UltraLite can also use execution plans to optimize queries. See “Execution plans in UltraLite” on page 446.

Remarks
Always remember to close the query. Otherwise memory cannot be freed and the number of temporary tables that remain can proliferate unnecessarily.

See also
- “UltraLite performance tips” on page 439
- “SELECT statement” [SQL Anywhere Server - SQL Reference]
- “Queries” [SQL Anywhere Server - SQL Usage]

Example
The following statement selects the number of employees from the Employees table.

```
SELECT COUNT(*)
FROM Employees
```

The following statement selects 10 rows from the Employees table starting from the 40th row and ending at the 49th row.

```
SELECT TOP 10 START AT 40 * FROM Employees
```

SET OPTION statement [UltraLite]
Changes the values of database options.

Syntax
```
SET OPTION option-name=[option-value]
```

- **option-name**: identifier
- **option-value**: string, identifier, or number
Remarks

This statement allows you to set options on an UltraLite database. Most UltraLite options are set when the database is initially created and cannot be modified afterward.

You cannot specify whether an option is persistent or not. UltraLite determines whether it is a persistent or temporary option. Persistent options are stored in the database. Temporary options are used only until the connection or database stops running.

UltraLite performs a commit when persistent options are set: \texttt{global\_database\_id} and \texttt{ml\_remote\_id}. UltraLite does not perform a commit on temporary or connection-based options.

The only database option that can be unset is \texttt{ml\_remote\_id}. For example:

\begin{verbatim}
SET OPTION ml\_remote\_id=
\end{verbatim}

The result is that the ID is set to NULL. When this occurs, UltraLite will automatically generate a new value at the next synchronization.

See also

- “UltraLite global\_database\_id option” on page 184
- “UltraLite ml\_remote\_id option” on page 185
- “UltraLite database options” on page 181
- “DB\_PROPERTY function [System]” on page 328

Example

The following statement sets the \texttt{global\_database\_id} option to 100:

\begin{verbatim}
SET OPTION global\_database\_id=100
\end{verbatim}

\textbf{START SYNCHRONIZATION DELETE statement [UltraLite]}

Restarts the logging of deleted rows for MobiLink synchronization.

Syntax

\begin{verbatim}
START SYNCHRONIZATION DELETE
\end{verbatim}

Remarks

UltraLite databases automatically log changes made to rows that need to be synchronized. UltraLite uploads these changes to the consolidated database during the next synchronization. This statement allows you to restart logging of deleted rows, previously stopped by a \texttt{STOP SYNCHRONIZATION DELETE} statement.

When a \texttt{STOP SYNCHRONIZATION DELETE} statement is executed, none of the delete operations executed on that connection are synchronized. The effect continues until a \texttt{START SYNCHRONIZATION DELETE} statement is executed.

Do not use \texttt{START SYNCHRONIZATION DELETE} if your application does not synchronize data.
The way in which UltraLite traces row states is unique. Be sure you understand the implication of deletes and row states.

See also

- “UltraLite database row state management” on page 453
- “STOP SYNCHRONIZATION DELETE statement [UltraLite]” on page 433

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION DELETE and STOP SYNCHRONIZATION DELETE.

```sql
STOP SYNCHRONIZATION DELETE;
DELETE FROM PROPOSAL
  WHERE last_modified < months( CURRENT TIMESTAMP, -1 );
START SYNCHRONIZATION DELETE;
COMMIT;
```

STOP SYNCHRONIZATION DELETE statement [UltraLite]

Stops the logging of deleted rows for MobiLink synchronization.

Syntax

```
STOP SYNCHRONIZATION DELETE
```

Remarks

UltraLite databases automatically log changes made to rows that need to be synchronized. UltraLite uploads these changes to the consolidated database during the next synchronization. This statement allows you to stop the logging of deleted rows, previously started using a STOP SYNCHRONIZATION DELETE statement. This command can be useful when making corrections to a remote database, but should be used with caution as it effectively disables MobiLink synchronization. You should only stop deletion logging temporarily.

When a STOP SYNCHRONIZATION DELETE statement is executed, no further delete operations executed on that connection are synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed.

Do not use STOP SYNCHRONIZATION DELETE if your application does not synchronize data.

The way in which UltraLite traces row states is unique. Be sure you understand the implication of deletes and row states.

See also

- “UltraLite database row state management” on page 453
- “START SYNCHRONIZATION DELETE statement [UltraLite]” on page 432

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION DELETE and STOP SYNCHRONIZATION DELETE.
STOP SYNCHRONIZATION DELETE;
DELETE FROM PROPOSAL
WHERE last_modified < months( CURRENT TIMESTAMP, -1 );
START SYNCHRONIZATION DELETE;
COMMIT;

SYNCHRONIZE statement [UltraLite]
Synchronize an UltraLite or UltraLite Java edition database via the MobiLink server.

Syntax

SYNCHRONIZE {
PROFILE sync-profile-name [ MERGE sync-option [ ;... ] ]
| USING sync-option [ ;... ]
}

sync-option :
 sync-option-name = sync-option-value

sync-option-name : string
 sync-option-value : string

Parameters

- sync-profile-name The name of the synchronization profile.

- MERGE clause Use this clause when you want to add or override options that are provided in the
  synchronization profile.

- USING clause Use this clause when you want to specify the synchronization options without
  referencing a synchronization profile.

- sync-option A string of one or more synchronization option value pairs, separated by semicolons.
  For example, 'option1=value1;option2=value2'.

- sync-option-name The name of the synchronization option.

- sync-option-value The value for the synchronization option.

Remarks

The synchronization is configured according to the parameters in the synchronization profile, or the
parameters can be specified in the statement itself.

By allowing synchronization options to be merged in, developers can choose to omit storing some options
in the database (like the MobiLinkPwd for instance).

If a synchronization callback function is defined and registered with UltraLite, whenever a
SYNCHRONIZE statement is executed, progress information for that synchronization is passed to the
callback function. If no callback is registered, progress information is suppressed.
Side effects

None.

See also

- “CREATE SYNCHRONIZATION PROFILE statement [UltraLite]” on page 410
- “ALTER SYNCHRONIZATION PROFILE statement [UltraLite]” on page 399
- “DROP SYNCHRONIZATION PROFILE statement [UltraLite]” on page 420
- “ULSetSynchronizationCallback method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]

Example

The following example shows the syntax for synchronizing a synchronization profile called Test1 where the MobiLinkPwd has not been stored as part of the profile:

```
SYNCHRONIZE PROFILE Test1 MERGE 'MobiLinkPwd=sql'
```

The following example shows the syntax for adding the publication and uploadonly options to a synchronization profile called Test1.

```
SYNCHRONIZE PROFILE Test1
    MERGE 'publications=p2;uploadonly=on'
```

The following example illustrates how to use USING.

```
SYNCHRONIZE USING
    'MobiLinkUid=joe;MobiLinkPwd=sql;ScriptVersion=1;Stream=TCPIP{host=localhost}
`

The following example shows the syntax for synchronizing the publication and uploadonly options.

```
SYNCHRONIZE
    USING 'publications=p2;uploadonly=on'
```

TRUNCATE TABLE statement [UltraLite]

Deletes all rows from a table without deleting the table.

Syntax

```
TRUNCATE TABLE table-name
```

Remarks

The TRUNCATE TABLE statement deletes all rows from a table and the MobiLink server is not informed of their removal upon subsequent synchronization. It is equivalent to executing the following statements:

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM TABLE;
START SYNCHRONIZATION DELETE;
```
Note
This statement should be used with great care on a database involved in synchronization or replication. Because the MobiLink server is not notified, this deletion can lead to inconsistencies that can cause synchronization or replication to fail.

After a TRUNCATE TABLE statement, the table structure, all the indexes, and the constraints and column definitions continue to exist; only data is deleted.

TRUNCATE TABLE cannot execute if a statement that affects the table is already being referenced by another request or query. Similarly, UltraLite does not process requests referencing the table while that table is being altered. Furthermore, you cannot execute TRUNCATE TABLE when the database includes active queries or uncommitted transactions.

For UltraLite.NET users: You cannot execute this statement unless you also call the Dispose method for all data objects (for example, ULDataReader). See “ULBulkCopy.Dispose method [UltraLite.NET]” [UltraLite - .NET Programming].

Schema changes Statements are not released if schema changes are initiated at the same time.

Side effects
If the table contains a column defined as DEFAULT AUTOINCREMENT or DEFAULT GLOBAL AUTOINCREMENT, TRUNCATE TABLE resets the next available value for the column.

Once rows are marked as deleted with TRUNCATE TABLE, they are no longer accessible to the user who performed this action, unless the user issues a ROLLBACK statement. However, they do remain accessible from other connections. Use COMMIT to make the deletion permanent, thereby making the data inaccessible from all connections.

If you synchronize the truncated table, all INSERT statements applied to the table take precedence over a TRUNCATE TABLE statement.

See also
● “UltraLite database schemas” on page 45
● “DELETE statement [UltraLite]” on page 418
● “START SYNCHRONIZATION DELETE statement [UltraLite]” on page 432
● “STOP SYNCHRONIZATION DELETE statement [UltraLite]” on page 433

Example
The following statement deletes all rows from the Departments table.

```
TRUNCATE TABLE Departments
```

If you execute this example, be sure to execute a ROLLBACK statement to revert your change.

**UNION statement [UltraLite]**

Combines the results of two or more select statements.
Note
This statement is not supported by UltraLite Java edition databases.

Syntax

```
select-statement-without-ordering
[ Union [ ALL | DISTINCT ] select-statement-without-ordering ]...
[ ORDER BY [ number [ ASC | DESC ], ... ]
```

Remarks

The results of several SELECT statements can be combined into a larger result using UNION. Each SELECT statement must have the same number of expressions in their respective SELECT list, and cannot contain an ORDER BY clause.

The results of UNION ALL are the combined results of the unioned SELECT statements. Specify UNION or UNION DISTINCT to get results without duplicate rows; however, removing duplicate rows adds to the total execution time for the statement. Specify UNION ALL to allow duplicate rows.

When attempting to combine corresponding expressions that are of different data types, UltraLite attempts find a data type in which to represent the combined values. If this is not possible, the union operation fails and an error is returned (for example “Cannot convert 'Surname' to a numeric”).

The column names displayed in the results are column names (or aliases) used for the first SELECT statement.

ORDER BY for UNION is restricted to the integer format. The ORDER BY clause uses integers to establish the ordering, where the integer indicates the query expression(s) on which to sort the results.

See also

- “SELECT statement [UltraLite]” on page 429

Example

The following example lists all distinct surnames found in the Employees and Customers tables, combined.

```
SELECT Surname FROM Employees
UNION
SELECT Surname FROM Customers
```

**UPDATE statement [UltraLite]**

Modifies rows in a table.

Syntax

```
UPDATE table-name[[AS] correlation-name]
SET column-name = expression, ...[
WHERE search-condition]
```
Parameters

- **table-name**: The *table-name* specifies the name of the table to update. Only a single table is allowed.

- **correlation-name**: An identifier to use when referencing the table from elsewhere in the statement.

**SET clause**

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. If the expression is a *column-name*, the old value is used.

Only columns specified in the SET clause have their values changed. In particular, you cannot use UPDATE to set a column's value to its default.

**WHERE clause**

If a WHERE clause is specified, only rows satisfying *search-condition* are updated.

Remarks

The UPDATE statement modifies values in a table.

Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive.

See also

- “INSERT statement [UltraLite]” on page 424
- “DELETE statement [UltraLite]” on page 418
- “Search conditions in UltraLite” on page 263

Example

The following statement transfers employee Philip Chin (employee 129) from the sales department to the marketing department (department 400).

```
UPDATE Employees
SET DepartmentID = 400
WHERE EmployeeID = 129
```

An example using *correlation-name*.

```
UPDATE Employee E
SET salary = salary * 1.05
WHERE EXISTS( SELECT 1 FROM Sales S HAVING E.Sales > Avg( S.sales) 
GROUP BY S.dept_no )
```
**UltraLite performance tips**

This section describes ways you can tune various UltraLite features to improve the performance of queries, inserts or updates.

**Cache size adjustment for an UltraLite database**

Although explicitly adjusting the cache size is not required, you typically want to adjust the cache size when your UltraLite database application is requested to reduce its memory usage by the operating system on mobile devices.

UltraLite database cache sizes increase dynamically in response to data operations and as available device memory allows within the parameters you specify. Normally you do not need to specify any parameters. If your database is large (400 MB for instance), you may want to specify the CACHE_MAX_SIZE parameter to raise the maximum limit beyond the default. UltraLite allocates some data structures based on the maximum cache size, so the default is not extremely large: you must explicitly request a large maximum to incur this extra memory overhead. There is no benefit to specifying a maximum cache size that is much larger than your maximum actual database file size.

UltraLite does not shrink the cache automatically. The database cache size can only be controlled explicitly in your application with the cache_allocation database option. In response to a low memory event raised by the operating system, adjust the cache_allocation database option after connecting to the database.

**Note**

Dynamic cache sizing is not supported by UltraLite Java edition databases. For more information about UltraLite Java edition database cache sizes, see “UltraLite Java edition database cache sizes” on page 440.

**Examples**

The following UltraLite C++ code sample illustrates how to set the maximum cache size to 100 MB by updating the connection string:

```cpp
static ul_char const * ConnectionParms =
  "UID=DBA;PWD=sql;DBF=sample.udb;CACHE_MAX_SIZE=100m";
```

The following UltraLite C++ code sample illustrates how to reduce the cache allocation in half to resize the cache:

```cpp
ULConnection * conn = ULDatabaseManager::OpenConnection(ConnectionParms);
ul_u_long percent;
percent = conn->GetDatabasePropertyInt( "cache_allocation" );
conn->SetDatabaseOptionInt( "cache_allocation", percent / 2 );
```
UltraLite Java edition database cache sizes

UltraLite databases are stored on a set of pages on all platforms. A working set of pages is maintained in cache and is managed using a first-in, first-out (FIFO) scheme. Pages that are currently in use are locked in cache to avoid being swapped out.

For larger databases, the database can be configured each time it is opened to limit the number of rows and index pages that resident in memory at one time.

You can graph the UltraLite Java edition database performance with benchmark tests to determine the optimal cache size for your business solution.

You can test different cache sizes and watch for performance to change abruptly. Your cache should be large enough to have a good working set of pages. Consider the following ideas to help you stress the cache:

- Create multiple indexes on the table and add foreign keys.
- Insert rows randomly (something other than the index order).
- Create large rows, at least 25% of the database page size.
- Set the index hash to something other than 0. This increased size also increases the page accesses needed.
- Start graphing performance based on the smallest cache size. For example, 256 KB on Windows (the smallest allowed cache for this platform) or 64 KB on all other platforms.
- Encrypt or obfuscate the database. Obfuscation uses less code compared to strong encryption and performs fewer computations. Simple encryption performance should only be marginally slower than no encryption at all. However, your security requirements ultimately dictate whether to use strong encryption.

See also
- “Index hashing” on page 441

Query performance tips

This section illustrates methods for improving the performance of executing queries in UltraLite databases.
Index scan creation and maintenance

You can create one or more indexes to improve the performance of your queries, or, depending on the type of index you create, to ensure that row values remain unique.

An index provides an ordering of a table’s rows based on the values in some or all of the columns. When creating indexes, the order in which you select columns to be indexed becomes the order in which the columns actually appear in the index. Indexes can greatly improve the performance of searches on the indexed column(s) when used strategically.

Use the following recommended practices for improving query performance:

- Create an index on any column:
  - for values that you search for on a regular basis
  - that the query uses to join tables
  - that are commonly used in ORDER BY, GROUP BY, or WHERE clauses

- Create a composite index and ensure that the first column of the index is used most often by the predicate in your query when creating it.

- Ensure that the update maintenance overhead an index introduces is not too high for the memory of your device.

- Do not create or maintain unnecessary indexes. Indexes must be updated when the data in a column is modified, so all insert, update, and delete operations are performed on the indexes as well.

- Create an index on large tables.

- Do not create redundant indexes. For example, if you create an index on table T with columns (x, y), you can create a redundancy if there is another existing index on T with columns (x, y, z).

See also

- “UltraLite indexes” on page 52
- “Manage temporary tables” on page 447
- “Direct page scans” on page 448
- “Viewing an execution plan” on page 449
- “About composite indexes” on page 53
- “EXPLANATION function [Miscellaneous]” on page 331
- “ULCommand.Plan property [UltraLite.NET]” [UltraLite - .NET Programming]
- “PreparedStatement.getPlan method [UltraLiteJ]” [UltraLite - Java Programming]

Index hashing

You can tune the performance of your queries by choosing a specific size for the maximum hash. A hash key represents the actual values of the indexed column. An index hash key aims to avoid the expensive operation of finding, loading, and then unpacking the rows to determine the indexed value. It prevents these operations by including enough of the actual row data with a row ID.
A row ID allows UltraLite to locate the actual row data in the database file. If you set the hash size to 0 (which disables index hashing), then the index entry only contains this row ID. If you set the hash size to anything other than 0, then a hash key is also used. A hash key can contain all or part of the transformed data in that row, and is stored with the row ID in the index page.

How much row data the hash key includes is determined by:

- The maximum hash size property you configure.
- How much is actually needed for the data type of the column.

**A hash example**

The value of an index hash maintains the order of the actual row data of indexed columns. For example, if you have indexed a LastName column for a table called Employees, you may see four names ordered as follows:

- Anders
- Anderseck
- Andersen
- Anderson

If you hashed the first six letters, your hash keys for these row values would appear as follows:

- Anders
- Anders
- Anders
- Anders

While these entries look the same, the first Anders in the list is used to represent the actual row value of **Anders**. The last Anders in the list, however, is used to represent the actual row value **Anderson**.

Consider the following statement:

```sql
SELECT *
FROM Employees
WHERE LastName = 'Andersen'
```

If the Employees table only contained a very high proportion of names similar to Andersen, then the hash key may not offer enough uniqueness to gain any performance benefits. In this case, UltraLite cannot determine if any of the hash keys actually meets the conditions of this statement. When duplicate index hash keys exist, UltraLite still needs to:

1. Find the table row that matches the row ID in question.
2. Load and then unpack the data so the value can be evaluated.
Performance benefits only occur when UltraLite can discern a proportionate number of unique hash so that the query condition evaluation is immediate to the index itself. For example, if the Employees table had thousands of names, there is still enough benefit to be gained by a hash of six letters. However, if the Employees table only contained an inordinate number of names that begin with Anders*, then you should hash at least seven letters so the degree of unique keys increases. Therefore, the original four names at the start of this example how are now represented with these hash keys:

- Anders
- Anderse
- Anderse
- Anderso

In this example, only two of the four row values would need to be unpacked and evaluated, rather than all four.

See also

- “Optimal hash size limit” on page 443
- “Adding an UltraLite index” on page 55
- “UltraLite max_hash_size creation parameter” on page 141

Optimal hash size limit

The UltraLite default maximum hash size of 4 bytes suits most deployments. You can increase the size to include more data with the row ID. However, this change could increase the size of the index and fragment it among multiple pages. This change can possibly increase the size of the database as a result. The impact of an increased maximum hash size depends on the number of rows in the table: for example, if you only have a few rows, a large index hash key would still fit on the index page. No index fragmentation occurs in this case.

When choosing an optimal hash size, consider the data type, the row data, and the database size (especially if a table contains many rows).

The only way to determine if you have chosen an optimal hash size is to run benchmark tests against your UltraLite client application on the target device. Observe how various hash sizes affect the application and query performance, in addition to the changes in database size itself.

Index hashing improves inserts, updates, deletes, and searches when the columns being indexed have a good distribution of values, such as strings that do not have a common prefix, at the cost of bigger index structures. Hashed indexes locate rows first by using the hash, and then by using direct row comparison to differentiate rows with the same hash value. If the hash size is sufficiently big, the hash uniquely identifies a row without reading and comparing the row. However, if the hash size is too big and the page size small, the index may need too many database pages.
The data type

To hash the entire value in a column, note the size required by each data type in the table that follows. UltraLite only uses the maximum hash size if it is necessary, and it never exceeds the maximum hash size you specify. UltraLite uses a smaller hash size if the column type does not use the full byte limit.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Bytes used to hash the entire value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONG VARCHAR, DOUBLE, FLOAT, REAL, LONG BINARY, ST_GEOMETRY</td>
<td>Not hashed.</td>
</tr>
<tr>
<td>BIT, TINYINT</td>
<td>1</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>INTEGER, DATE</td>
<td>4</td>
</tr>
<tr>
<td>BIGINT, DATETIME, TIME, TIMESTAMP, TIMESTAMP WITH TIMEZONE</td>
<td>8</td>
</tr>
<tr>
<td>DECIMAL, NUMERIC</td>
<td>Approximately the precision divided by two.</td>
</tr>
<tr>
<td>CHAR, VARCHAR</td>
<td>To hash the entire string, the maximum hash size in bytes must match the declared size of the column. In a UTF-8 encoded database, always multiply the declared size by a factor of 2, but only to the allowed maximum of 32 bytes. For example, if you declare a column VARCHAR(10) in a non-UTF-8 encoded database, the required size is 10 bytes. However, if you declare the same column in a UTF-8 encoded database, the size used to hash the entire string is 20 bytes. For UltraLite Java edition databases, each VARCHAR character is hashed using its UTF-8 representation, which may be 1-3 characters long. UltraLite Java edition uses (3 * n) bytes to hash a VARCHAR(n) column. However, you may specify a hash size of n bytes to hash all values if the VARCHAR(n) column only contains ASCII7 characters.</td>
</tr>
<tr>
<td>BINARY, VARBINARY</td>
<td>The maximum hash size in bytes must match the declared size of the column. For example, if you declare a column BINARY(30), the required size is 30 bytes.</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>16</td>
</tr>
</tbody>
</table>
For example, if you set a maximum hash size of 6 bytes for a two-column composite index that you declared as INTEGER and BINARY (20) respectively, then based on the data type size requirements, the following changes occur:

- The entire value of the row in the INTEGER column is hashed and stored in the index because only 4 bytes are required to hash integer data types.
- Only the first 2 bytes of the BINARY column are hashed and stored in the index because the first 4 bytes are used by the INTEGER column. If these remaining 2 bytes do not hash an appropriate amount of the BINARY column, increase the maximum hash size.

The row data
The row values of the data being stored in the database also influence the effectiveness of a hashed index.

For example, if you have a common prefix shared among entries of a given column, you can render the hash ineffective if you choose a size that only hashes prefixes. In this case, choose a size that ensures more than just the common prefix is hashed. If the common prefix is long, consider not hashing the values at all.

When a non-unique index stores many duplicate values, and UltraLite cannot hash the entire value, the hash likely cannot improve performance.

The database size
Each index page has some fixed overhead, but the majority of the page space is used by the actual index entries. A larger hash size means each index entry is bigger, which means that fewer entries can fit on a page. For large tables, indexes with large hashes use more pages than indexes with small or no hashes. Requiring more pages increases the database size and degrades performance. The latter typically occurs because the cache can only hold a fixed number of pages, thereby causing UltraLite to swap pages.

The following table gives you an approximation of how the hash size can affect the number of pages required to store data in an index:

<table>
<thead>
<tr>
<th>Table</th>
<th>Page size</th>
<th>Hash size</th>
<th>Number of entries</th>
<th>Pages required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table A</td>
<td>4 KB</td>
<td>0</td>
<td>1200</td>
<td>3 pages</td>
</tr>
<tr>
<td>Table B</td>
<td>4 KB</td>
<td>32 bytes</td>
<td>116</td>
<td>3 pages</td>
</tr>
<tr>
<td>Table C</td>
<td>4 KB</td>
<td>32 bytes</td>
<td>1200 entries</td>
<td>11 pages</td>
</tr>
</tbody>
</table>

Set the hash size
You can set the maximum hash size in two ways:

- To store a database default for the maximum size, set the max_hash_size creation parameter when you create your database. If you do not want to hash indexes by default, set this value to 0. Otherwise, you can change it to any value up to 32 bytes, or keep the UltraLite default of 4 bytes.
- Override the default by setting a specific hash size when you create a new index. Use one of the following approaches:
In Sybase Central, set the Maximum Hash Size property when creating a new index.

With SQL, use the WITH MAX HASH SIZE clause in either the CREATE TABLE or CREATE INDEX statement.

See also
- “Adding an UltraLite index” on page 55
- “UltraLite SQL data types” on page 275
- “UltraLite max_hash_size creation parameter” on page 141
- “CREATE INDEX statement [UltraLite]” on page 407
- “CREATE TABLE statement [UltraLite]” on page 411

Execution plans in UltraLite

UltraLite execution plans show how tables and indexes are accessed when a query is executed. UltraLite includes a query optimizer. The optimizer is an internal component of the UltraLite runtime that attempts to produce an efficient plan for the query. It tries to avoid the use of temporary tables to store intermediate results and attempts to ensure that only the pertinent subset of a table is accessed when a query joins two tables.

Overriding the optimizer

The optimizer always aims identify the most efficient access plan possible, but this goal is not guaranteed—especially with a complicated query where a great number of possibilities may exist. In extreme cases, you can override the table order it selects by adding the OPTION (FORCE ORDER) clause to a query, which forces UltraLite to access the tables in the order they appear in the query. This option is not recommended for general use. If performance is slow, a better approach is usually to create appropriate indexes to speed up execution.

Tip
If you are not going to update data with the query, you should specify the FOR READ ONLY clause in your query. This clause may yield better performance. See “SELECT statement [UltraLite]” on page 429.

Determining the access method used by the optimizer

The UltraLite optimizer uses sophisticated optimization strategies when choosing an index for query optimization. However, with simple queries you cannot easily predetermine which index the optimizer uses to optimize the query performance, or if an index is used at all. As the complexity increases, the index selected depends on the clauses required by your query. Usually, the presence of a FOR READ ONLY clause may cause the optimizer to choose a direct table scan instead of an index to yield better query performance.

When optimizing a query, the optimizer looks at the requirements of the query and checks if there are any indexes that it can use to improve performance. If performance cannot be improved with any index, then the optimizer does not scan one: either a temporary table or a direct page scan is used instead. Therefore,
you may need to experiment with your indexes and frequently check the generated execution plans to ensure that:

- You are not maintaining indexes that are not being used by the optimizer.
- You are minimizing the number of temporary tables being created.

For complex queries, knowing which index is used is even less predictable. For example, when a query contains a WHERE predicate and a GROUP BY clause in addition to an ORDER BY clause, one index alone might not satisfy the search conditions of this query. So, if you have created an index to meet the selectivity requirements of the WHERE predicate, you may find that the optimizer does not actually use it. Instead, the optimizer may use an index that offers better performance for the ORDER BY conditions because this clause could require the most processing.

Checking the execution plan

You can check the execution plan either programmatically with the appropriate API call or in the Plan Viewer in Interactive SQL:

- **If no index is used** the execution plan appears as follows:

  \[\text{scan}(T)\]

- **If a temporary table is used** the execution plan appears as follows:

  \[\text{temp} \ [\text{scan}(T)]\]

- **If an index is used** the index name is included the execution plan:

  \[\text{scan} \ (T, \ index\_name)\]

See also

- “Manage temporary tables” on page 447

Manage temporary tables

In general, the optimizer always tries to avoid creating temporary tables to return query results because the entire temporary table must be populated before the first row can be returned. If an index exists, the optimizer tries to use the index first and only creates a temporary table as a last resort.

A temporary table is used by an access plan to store data during its execution in a transient or temporary work table. This table only exists while the access plan is being executed. Generally, temporary tables are used when intermediate results do not fit in the available memory, such as:

- When subqueries need to be evaluated early in the access plan.
- When data in a temporary table is held for a single connection only.
- When a query contains an ORDER BY on a column other than an index.
- When a query contains a GROUP BY on a column other than an index.
It is difficult to anticipate whether an index you have created avoids the necessity for a temporary table. Therefore, check the plans for a query to ensure the indexes you have created are actually being used by the UltraLite query optimizer.

You can avoid using temporary tables by using an index for the columns used in the ORDER BY or GROUP BY clauses.

**See also**

- “UltraLite TEMP_DIR connection parameter” on page 175
- “Determining the access method used by the optimizer” on page 446
- “How to read execution plans” on page 450
- “When to view execution plans” on page 448

**Direct page scans**

UltraLite uses direct page scans as an alternative to index scans when it is more efficient to access information directly from the database page. A direct page scan is only used after the optimizer confirms that:

- No preexisting index can return results more efficiently.
- You are not using the query to perform updates. For example, you have declared the statement to be FOR READ ONLY (the default setting if no FOR clause has been specified), or have written the query in such a way that it is obvious that data is not being updated.

Because UltraLite reads the rows directly from the pages on which the rows are stored, query results are returned without order. The order of subsequent query results is unpredictable. If you need the order of rows to be predictable and deterministic, use an ORDER BY clause to get results in a consistent order. However, if order is not important, you can omit the ORDER BY clause to improve query performance.

**Note**

You cannot use direct page scans if you are using a ULTable class in an UltraLite API to program your application.

You can check to see when UltraLite scans a page directly or which index was used to return results.

**See also**

- “Determining the access method used by the optimizer” on page 446
- “Index scan creation and maintenance” on page 441

**When to view execution plans**

View an execution plan in Interactive SQL when you need to know:

- What index will be used to return the results. An index scan object contains the name of the table and the index on that table that is being used.
Whether a temporary table will be used to return the results. Temporary tables are written to the UltraLite temporary file.

Which order tables are joined. This information allows you to determine how performance is affected.

Why a query is running slowly or to ensure that a query does not run slowly.

See also

“UltraLite TEMP_DIR connection parameter” on page 175

Viewing an execution plan

Use Interactive SQL to display an UltraLite plan that summarizes how a prepared statement is to be executed. The text plan is displayed in the Interactive SQL Plan Viewer.

Prerequisites

There are no prerequisites for this task.

Context and remarks

In UltraLite, an execution plan is strictly a short textual summary of the plan. No other plan types are supported. However, being a short plan, it allows you to compare plans quickly, because information is summarized on a single line.

Task

1. Click Tools » Plan Viewer.
2. In the SQL pane, type a query.
3. Click Get Plan to generate a plan for the specified SQL statements.

Results

The text plan appears in the lower pane of the Plan Viewer.

Example

Consider the following statement:

```sql
SELECT I.inv_no, I.name, T.quantity, T.prod_no
FROM Invoice I, Transactions T
WHERE I.inv_no = T.inv_no
```

This statement might produce the following plan:

```
join[scan(Invoice,primary),index-scan(Transactions,secondary)]
```
The plan indicates that the join operation is completed by reading all rows from the Invoice table (following an index named primary). It then uses the index named secondary from the Transactions table to read only the row whose inv_no column matches.

See also
- “Interactive SQL utility (dbisql)” [SQL Anywhere Server - Database Administration]
- “How to read execution plans” on page 450

How to read execution plans
Because UltraLite short plans are textual summaries of how a query is accessed, you need to understand how the operations of either a join or a scan of a table are implemented.

- **For scan operations** Represented with a single operand, which applies to a single table only and uses an index. The table name and index name are displayed as round brackets ( (, ) ) following the operation name.

- **For other operations** Represented with one or more operands, which can also be plans in and of themselves. In UltraLite, these operands are comma-separated lists contained by square brackets ( [ ] ).

Operation list
Operations supported by UltraLite are listed in the table that follows.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count(*)</td>
<td>Counts the number of rows in a table.</td>
</tr>
<tr>
<td>distinct[ plan ]</td>
<td>Implements the DISTINCT aspect of a query to compare and eliminate duplicate rows. It is used when the underlying plan sorts rows in such a way that duplicate contiguous rows are eliminated. If two contiguous rows match, only the first row is added to the result set.</td>
</tr>
<tr>
<td>dummy</td>
<td>No operation performed. It only occurs in two cases:</td>
</tr>
<tr>
<td></td>
<td>- When you specify DUMMY in a FROM clause.</td>
</tr>
<tr>
<td></td>
<td>- When the FROM clause is missing from the query.</td>
</tr>
<tr>
<td>filter[ plan ]</td>
<td>Executes a search condition for each row supplied by the underlying plan. Only the rows that evaluate to true are forwarded as part of the result set.</td>
</tr>
<tr>
<td>group-by[ plan ]</td>
<td>Creates an aggregate of GROUP BY results, to sort multiple rows of grouped data. Rows are listed in the order they occur and are grouped by comparing contiguous rows.</td>
</tr>
</tbody>
</table>
### Operation Description

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group-single[ plan ]</td>
<td>Creates an aggregate of GROUP BY results, but only when it is known that a single row will be returned.</td>
</tr>
<tr>
<td>keyset[ plan ]</td>
<td>Records which rows were used to create rows in a temporary table so UltraLite can update the original rows. If you do not want those rows to be updated, then use the FOR READ ONLY clause in the query to eliminate this operation.</td>
</tr>
<tr>
<td>index-scan( table-name, index-name )</td>
<td>Reads only part of the table; the index is used to find the starting row.</td>
</tr>
<tr>
<td>join[ plan, plan ]</td>
<td>Performs an inner join between two plans.</td>
</tr>
<tr>
<td>lojoin[ plan, plan ]</td>
<td>Performs a left outer join between two plans.</td>
</tr>
<tr>
<td>like-scan( table-name, index-name )</td>
<td>Reads only part of a table; the index is used to find the starting row by pattern matching.</td>
</tr>
<tr>
<td>rowlimit[ plan ]</td>
<td>Performs the row limiting operation on propagated rows. Row limits are set by the TOP n or FIRST clause of the SELECT statement.</td>
</tr>
<tr>
<td>scan( table-name, index-name )</td>
<td>Reads an entire table following the order indicated by the index.</td>
</tr>
<tr>
<td>sub-query[ plan ]</td>
<td>Marks the start of a subquery.</td>
</tr>
<tr>
<td>temp[ plan ]</td>
<td>Creates a temporary table from the rows in the underlying plan. UltraLite uses a temporary table when underlying rows must be ordered and no index was found to do this ordering. You can add an index to eliminate the need for a temporary table. However, each additional index used increases the duration needed to insert or synchronize rows in the table for which the index applies.</td>
</tr>
<tr>
<td>union-all[ plan, ..., plan ]</td>
<td>Performs a UNION ALL operation on the rows generated in the underlying plan.</td>
</tr>
</tbody>
</table>

## Insert and Update performance tips

This section illustrates methods for improving the performance of executing data inserts and updates in UltraLite databases.
Transaction and row state management

UltraLite maintains state information along with the data in the database. UltraLite tracks and stores state information so it can manage:

● Concurrent connections, so UltraLite can share resources as required.

● Synchronization progress state, to ensure that synchronization occurs successfully.

● Row state, to maintain data integrity by tracking how data has changed between synchronizations.

● Transactions, to determine when and how data gets committed. In UltraLite, a transaction is processed in its entirety or not at all.

● Recovery and backup information, to protect data against operating system crashes, and end-user actions such as removing storage cards, or device resets while UltraLite is running.

See also

● “UltraLite concurrency” on page 452
● “UltraLite synchronization client features” on page 2
● “UltraLite database row state management” on page 453
● “UltraLite transaction processing” on page 454
● “UltraLite and UltraLite Java edition database back up and recovery” on page 42

UltraLite concurrency

UltraLite automatically uses the following methods to manage concurrent database accesses:

● **Multiple UltraLite database access** A single application can open connections to multiple databases. UltraLite Java edition does not support simultaneous access.

● **Multiple applications** An UltraLite or UltraLite Java edition database can only be opened by one process at a time.

● **Multiple threads** UltraLite supports multi-threaded applications. A single application can be written to use multiple threads, each of which can connect to the same or different databases.

● **Multiple transactions/requests** Each connection can have a single transaction in progress at one time. Transactions can consist of a single request or multiple requests. Data modifications made during a transaction are not made permanent in the database until the transaction is committed. Either all data modifications made in a transaction are committed, or all are rolled back.

● **Synchronization** During upload and download, read-write access to the database is permitted. However, if an application changes a row that the download then attempts to change, the download fails and rolls back. Use the Disable Concurrency synchronization parameter to disable access to data during synchronization.

If synchronization fails, UltraLite supports resumable downloads on all platforms.
See also

- “Failed downloads” [MobiLink - Server Administration]
- “UltraLite and UltraLite Java edition database limitations” on page 8
- “UltraLite transaction processing” on page 454
- “UltraLite clients” on page 63
- “Additional Parameters synchronization parameter” on page 83

UltraLite database row state management

Maintaining row state information is a powerful part of the UltraLite database feature set. Tracking the state of tables and rows is particularly important for data synchronization.

Note

UltraLite Java edition does not support row state management. UltraLite Java edition databases use transaction logs to track changes that need to be synchronized.

An internal marker is used to keep track of the row state in an UltraLite database. Row states control transaction processing, recovery, and synchronization. When an application inserts, updates, or deletes a row, UltraLite modifies the state of the row to reflect the operation and the connection that performed the operation. When a transaction is committed, the states of all rows affected by the transaction are modified to reflect the commit. If an unexpected failure occurs during a commit, the entire transaction is rolled back. The following list summarizes these behaviors:

- **When a delete is issued**  The state of each affected row is changed to reflect the fact that it was deleted. When a delete is undone through a rollback, the original state of the row is restored.

- **When a delete is committed**  The affected rows are not always removed from memory. If the row has never been synchronized, then it is removed. If the row has been synchronized, then it is not removed, because the delete operation needs to be synchronized to the consolidated database first. After the next synchronization, the row is removed from memory.

- **When a row is updated**  A new version of the row is created. The states of the old and new rows are set so the old row is no longer visible and the new row is visible.

- **When a row update is committed**  When a transaction is committed, the states of all rows affected by the transaction are modified to reflect the commit. When an update is synchronized, both the old and new versions of the row are needed to allow conflict detection and resolution. The old row is then deleted from the database and the new row simply becomes a normal row.

- **When a row is added**  The row is added to the database and is marked as not committed.

- **When an added row is committed**  The row is marked as committed and is also flagged as requiring synchronization with the consolidated database.
UltraLite transaction processing

A transaction is a logical set of operations that are executed atomically: either all operations in the transaction are stored in the database or none are. An UltraLite application's access to the UltraLite runtime is serialized. While it is possible for multiple transactions to be open simultaneously, UltraLite only processes one transaction at a time. This behavior means that an application cannot:

- Have blocked transactions (also known as deadlocks). UltraLite never blocks a request based on an existing row lock. In this case, UltraLite immediately returns an error.

- Overwrite outstanding changes. A transaction cannot overwrite another transaction's outstanding changes. When a transaction changes a row, UltraLite locks that row until the transaction is committed or rolled back. The lock prevents other transactions from changing the row, although they can still read the row.

Tip

All UltraLite APIs—except the UltraLiteJ and C++ APIs—can operate in autocommit mode.

In autocommit mode, UltraLite executes a commit after each operation. Some APIs use autocommit by default. If you are using one of these interfaces, you must set autocommit to off to exploit multi-operation transactions. The way of turning autocommit off depends on the programming interface you are using. In most interfaces it is a property of the connection object.

See:

- UltraLite C++: “Transaction management” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Transaction management” [UltraLite - .NET Programming]
- UltraLiteJ: “Row operation management” [UltraLite - Java Programming]

For example, two applications, A and B, are reading the same row from the database and they both calculate new values for one of its columns based on the data they read. If A updates the row with its new value and B then tries to modify the same row, B gets an error. An attempt to change a locked row sets the error SQLCODE SQLE_LOCKED, while an attempt to change a deleted row sets the error SQLCODE SQLE_NOTFOUND. Therefore, you should program your application so it checks the SQLCODE value after attempting to modify data.

See also

- UltraLite C++: “Error handling” [UltraLite - C and C++ Programming]
- UltraLite.NET: “Error handling” [UltraLite - .NET Programming]
- UltraLiteJ: “Error handling” [UltraLite - Java Programming]
Flush single or grouped transactions

You can choose your recovery point in UltraLite by delaying committed transaction flushes. When UltraLite releases the commit to storage, the recovery point helps control when a subset of SQL statements in a transaction triggers additional operational overhead.

By default, UltraLite uses an operational-based default that flushes individual transactions to storage immediately upon a commit. For some deployments, these frequent operations can be excessive and limit the amount of transaction throughput. To reduce the performance expense caused by this default, you may choose a state-based approach. Especially for applications that rely on autocommit operations, this approach delays the additional overhead required to flush the committed transactions to storage:

- **On checkpoint**  You can set your own checkpoint, and then use it to release the work performed over the course of time. You can use as many checkpoints as you require, either within a single transaction or over multiple transactions.

- **Grouped**  You can choose a transaction count threshold and/or a timeout threshold to release the work performed.

Delaying commit flushes based on state yields better performance and a cleaner application design because applications are not required to wait for a response from UltraLite. By delaying commit flushes you also minimize the exposure to transactions by giving more granular control over data for which work has not been fully completed. For example, in a sales application, an order may be available to a second application before all items have been added or even approved.

However, it is important for you to take into account the recoverability of a transaction for which commit flushes have been delayed. Transactions that have not been released cannot be recovered. Therefore, you need to evaluate the trade-off between the data integrity of your application and its performance.

**See also**
- “UltraLite COMMIT_FLUSH connection parameter” on page 163
- “UltraLite commit_flush_count option [temporary]” on page 182
- “UltraLite commit_flush_timeout option [temporary]” on page 183
- “CHECKPOINT statement [UltraLite]” on page 405
- “ULCheckpoint method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULConnection.Checkpoint method [UltraLite C++]” [UltraLite - C and C++ Programming]

**UltraLite benchmark tips**

Benchmark testing activity is generally performed before reaching the production stage of the application development cycle. This phase requires that you test the UltraLite database with the application and ensure both components inter-operate as efficiently as possible. If you discover through your tests that performance is not as efficient as it could be, you can then tune your database and/or optimize your application to improve benchmark results.
Note
If your UltraLite deployment is part of a MobiLink synchronization environment, remember to test synchronization performance as well. See “MobiLink tuning for performance” [MobiLink - Server Administration].

Types of benchmark tests

You can run benchmark tests to check the performance of:

- SQL statements
  The UltraLite database has been optimized to handle SQL queries efficiently and return results quickly. Nonetheless, you should see how well important queries perform to improve database performance.

- Synchronization
  The key to achieving optimal MobiLink synchronization throughput is to have multiple synchronizations occurring simultaneously and executing efficiently.

- Indexes

- Table design

- Application code

- Device configuration
  For example, compare using external flash memory and a device's internal memory as deployment locations for UltraLite.

- Database configuration
  For example, try different cache sizes, page sizes, reserve sizes, indexes, hash sizes, and so on.

- Data throughput (based on transactions per second)
  While UltraLite is not typically a database intended for mass data entry processing, depending on your business requirements, this benchmark is one that you may to run tests for.

- Software changes
  You should test the effect of software changes between two different versions of UltraLite or different versions of an application.

See also

- “SQL query testing” on page 457
- “MobiLink tuning for performance” [MobiLink - Server Administration]
SQL query testing

SQL queries can range from simple to complex. Depending on the nature and importance of your queries, there are two types of query benchmark testing available to you.

Representational SQL benchmarks

This type of testing requires that you test a selection of statements that are representative of typical transactions that the application performs during day-to-day operations. Different applications require different benchmark tests because the fundamental business use for each application varies. For example, a meter reading application might simply test a basic INSERT statement. However a mobile sales force application might test multiple INSERT statements, in addition to multiple SELECT statements and perhaps an UPDATE statement.

The volume of queries in your application can limit what you can realistically test. If you have excessive query processing, you may be limited to performing the targeted SQL benchmark testing.

Targeted SQL benchmarks

If you have a lot of statements used by your applications, you may want to narrow the scope of your test to include some or all of the following:

- The most frequently-used statements.
- The statements that process high volumes of data.
- The statements that have time-sensitive requirements.
- The statements that are most important to the business case of your application.
- The most complex statements: for example, those that have the largest number of table joins or that use many subqueries. These types of statements can use a large amount of device resources. Even if the statements are only used infrequently, you may want to check that they don't exceed the capacity of the device.
- The statements that are not supported by an index.
- The statements that use a large amount of memory resources.

Methodology

1. The preparation phase

   Allows you to finalize your database design and reach a stable point in your application development before starting your benchmark testing.

2. The creation phase

   Allows you to build a custom program that replicates the end-user behavior you predict for your UltraLite deployment.
3. **The execution and analysis phase**

   Allows you to fine tune different elements of your database and record the results of those changes so you can analyze them. Tests are repeated until the maximum benefit of all modifications has been reached.

---

**The preparation phase**

The preparation phase allows you to get your database and application in a state where they can be successful benchmark candidates, and to determine what goals you hope to achieve from your tests. Do the following:

1. Complete the logical design of the database.
   
   Ensure you have:
   
   - Created and populated tables with representative data.
   - Created indexes to retrieve data in those tables more effectively.

2. Prepare the physical deployment environment of both the database and application. The deployment environment must accurately represent the final production environment: that is, the lab and production environments should share the same memory and disk configurations on the same platform/device type.

3. Ensure you have reached a stable point in your application programming phase. Remember you are looking for performance optimizations not defects; however, the latter may also be revealed as a result of testing.

   All database queries should access and return as much data as required. If the production environment requires sorting of data, ensure queries include this data as well. Otherwise, the application cannot accurately test representative memory requirements.

4. Deploy a copy of the database and application to the final disk location.

5. Decide what element of the database performance you want to check and potentially tune.

   You can now run benchmark tests against your database and application.

---

**See also**

- “Types of benchmark tests” on page 456
- “Methodology” on page 457

---

**The creation phase**

You must create tests that yield reliable results. Otherwise, you cannot legitimately compare results over time.

The following characteristics make a benchmark test effective and reliable:
● **Goal**  Are you looking to capture a performance ratio or are you trying to see the duration required to process a command against a database? For the former, if you are testing SQL performance, you may want to run one or more statements repeatedly until a set time interval has expired. This testing gives you the throughput ratio, which can be summarized as follows:

\[
\text{statement-number} / \text{time-interval} = \text{throughput ratio}
\]

● **Environment**  Establish a test environment as your baseline and record the design and scope of it. If you cannot run the same test under the same conditions, you cannot legitimately compare results of that test. Additionally, the hardware and software you use in the lab as part of your benchmark test should match that of your production environment.

● **State**  Reliable benchmark tests always start each iteration with the same action. Decide whether or not third-party applications should operate concurrently with UltraLite. If they affect performance, you should add them to the benchmark test design. For third-party applications that should not be running, always exit these applications completely—even minimized or idle applications/processes could skew results because memory is still being used.

● **Results**  Results of benchmarks must be captured in a consistent way after each iteration of the test. Over time the results can indicate a trend and help you determine what changes can yield an improvement in UltraLite performance—either in the database or in the application (or both).

● **Timing mechanism**  Benchmark tests simulate user actions; therefore, you typically track the elapsed execution times of these actions. Ensure your timing mechanism is systematic so execution times are accurately reflected in the results of your tests.

**See also**

- “Methodology” on page 457
- “SQL query testing” on page 457

**The execution and analysis phase**

Benchmark execution is the phase during which you tune your database by iteratively running a test, modifying something about the database (for example, the value of one or more database properties or connection parameters), and then running the test again to see the outcome of any changes.

**Prerequisites**

There are no prerequisites for this task.

**Context and remarks**

The following procedure assumes you are testing different database properties and/or connection parameters to find the maximum benefit. Repeat this procedure until all parameters that require testing have been tested.

<table>
<thead>
<tr>
<th><strong>Tip</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose only those properties or parameters that are significant to the workload and the objectives of your UltraLite deployment.</td>
</tr>
</tbody>
</table>
Task

1. Create a baseline by running the first iteration of the test. In this case, because you are testing different database properties and/or connection parameters, you would use UltraLite defaults wherever possible.

2. Begin your normal test runs by tuning only one database property or connection parameter at a time. This limitation ensures that the results you collect are systematic in their approach and helps you more readily determine when you have reached the maximum benefit of your tuning activities.

3. Output from the benchmark program should include:
   - an identifier or label for each test
   - the iteration of the program execution
   - the name of the element being checked and what you did to change it
   - the recorded elapsed time

For example, even though you could test other database parameters, if you limited your test to just varying page sizes, cache sizes, and reserve sizes, your output might be saved to a table that looks similar to the example that follows:

<table>
<thead>
<tr>
<th>PROP/PARM</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST NUMBER</td>
<td>001</td>
</tr>
<tr>
<td>page_size</td>
<td>1</td>
</tr>
<tr>
<td>CACHE_SIZE</td>
<td>128</td>
</tr>
<tr>
<td>RESERVE_SIZE</td>
<td>128</td>
</tr>
<tr>
<td>STMT ID</td>
<td>EXECUTION (seconds)</td>
</tr>
<tr>
<td>01</td>
<td>01.55</td>
</tr>
<tr>
<td>02</td>
<td>02.01</td>
</tr>
<tr>
<td>03</td>
<td>00.33</td>
</tr>
</tbody>
</table>

4. When you have completed an iteration, return the database to the baseline state to ensure you do not inadvertently contaminate results of subsequent runs.

Results

Depending on the results of the benchmark test, do one of the following:

- If performance improves, change the value of the same property or parameter and rerun the test. Keep tuning this value until you cannot improve performance any further.

- If the performance worsens, return the value of the property or parameter to the previous value.

Next

Test another new property or parameter.

See also

- “Methodology” on page 457
UltraLite troubleshooting

Unable to start the UltraLite engine

Symptom
You have to use the START connection parameter to start the UltraLite engine with the following definition; however, the client returns SQLE_UNABLE_TO_CONNECT_OR_START.

START="\Program Files\uleng16.exe"

Explanation
The location of the quotes is incorrect.

Recommendation
For this parameter to work, the first quotation mark must follow the \ character. For example, you can delimit spaces in this path as follows:

START="\Program Files\uleng16.exe"

or

START='\Program Files\uleng16.exe'"

Unable to connect to databases after upgrade

Symptom
You have upgraded UltraLite. You discover that you are able to create an empty database using the administration tools. However, when you try to connect to this or any other database (including CustDB.udb) with Sybase Central, you receive an error. Connecting to SQL Anywhere databases works without incident, however.

Explanation
You did not close all SQL Anywhere applications and processes. Therefore, your UltraLite plug-ins were not installed correctly.

Recommendation
Remove and reinstall SQL Anywhere.

1. Close Sybase Central, Interactive SQL, and any running database engines.

2. Run the following commands:

dbsql -terminate

scjview -terminate
3. Open the Windows Task Manager, and end any scjview.exe and dbisql.exe processes.

4. Reinstall the latest version of UltraLite.

See also
● “UltraLite upgrades” [SQL Anywhere 16 - Changes and Upgrading]

UltraLite database corruption

Symptom
Your UltraLite database may be corrupt if it:

● Generates the following errors:
   ○ SQLE_DEVICE_ERROR
   ○ SQLE_DATABASE_ERROR (can also be a symptom of other issues)
   ○ SQLE_MEMORY_ERROR (can also be a symptom of other issues)

● Crashes or returns invalid query results.

Explanation
There are two more typical causes corruption:

● The more frequent cause occurs if the device has problems storing the file, thereby spuriously changing the contents of it. This issue usually stops the UltraLite database from functioning fairly quickly.

● The less frequent cause occurs if an error in the UltraLite code fails to maintain an index correctly. These issues can go undetected for much longer because the change to the results of a query are more subtle.

Recommendation
Checksums are used to detect offline corruption in an UltraLite database, which can help reduce the chances of other data being corrupted as the result of a bad critical page. If a checksum validation fails when the UltraLite database loads a page, UltraLite immediately stops the database and reports a fatal error. This error cannot be corrected. Instead you must:

1. Report the error. It is helpful if you know the sequence of events that caused the corruption to occur, and if the error is reproducible.

2. If you need the data, unload the contents of the UltraLite database to a file.

3. Create a new UltraLite database.

4. Repopulate the data either by synchronizing or by loading the unloaded data.

See also
● “UltraLite checksum_level creation parameter” on page 134
**Database size not stabilizing**

**Symptom**
Your application collects a lot of large binary objects among multiple client devices, synchronizes this information to a consolidated database, and then the synchronized data is deleted from each client device. However, the database size remains large despite the data being removed from the database. This is a concern because file size needs to be managed carefully due to limited resources of the device.

**Explanation**
Database size should only increase if your data grows in the database. However, once grown, the database file keeps that size, and does not shrink on its own. Free space is maintained internally to the file.

**Recommendation**
Ensure you are not using the STOP SYNCHRONIZATION DELETE or TRUNCATE statements for tables that do not get synchronized. Instead use the DELETE statement with a FROM `table-name` clause for tables that do not get synchronized.

Recreate the database post-synchronization:

1. Create the database that is deployed to the devices.
2. Creating a SQL script of DDL statements that define the schema required by the client devices. See “Deploying UltraLite database schema upgrades” on page 115.
3. Synchronize the data.
4. Drop the database.
5. Create a new, empty database and use standard database schema with the ALTER DATABASE SCHEMA FROM FILE statement.

**See also**
- “STOP SYNCHRONIZATION DELETE statement [UltraLite]” on page 433
- “TRUNCATE TABLE statement [UltraLite]” on page 435
- “DELETE statement [UltraLite]” on page 418
- “ALTER DATABASE SCHEMA FROM FILE statement [UltraLite]” on page 397

**Importing ASCII data into a new UltraLite database**

**Symptom**
You have created a new UltraLite database, but have a `.csv` ASCII data file that you cannot import.

**Explanation**
The `.csv` format is not supported by any of the UltraLite administration tools.
Recommendation

You can try one of the following techniques:

- Use Interactive SQL (dbisql) to import the data. You can connect to the UltraLite database and then click Data » Import Data. Alternatively, you can connect to the UltraLite database and run the INPUT statement (this statement cannot be used in an UltraLite PreparedStatement object).

  **Note**
  
  UltraLite requires primary keys. Although Interactive SQL can create the table for you, it does not automatically create the primary keys for them. Always connect to an empty UltraLite database you have created for this purpose.

- If you incorporate this functionality as part of a batch process, you must write your own code.

See also

- “INPUT statement [Interactive SQL]” [SQL Anywhere Server - SQL Reference]
- “Interactive SQL for UltraLite utility (dbisql)” on page 187

Utilities still running as the previous version

Symptom

You have just installed UltraLite 16. However, when you try to run any of the UltraLite utilities, the previous version starts.

Explanation

If you have multiple versions of UltraLite on your computer, you must pay attention to your system path when using the administration. Since the installation adds the most recently installed version executable directory to the end of your system path, it is possible to install a new version of the software, and still inadvertently be running the previously installed version.

Recommendation

There are various workarounds to this problem.

See also

- “How to ensure that you are running the correct version of the utilities when you have multiple versions installed” [SQL Anywhere 16 - Changes and Upgrading]

Result set changes unpredictably

Symptom

You run a query and the result set you expect changes each time you run it.
**Explanation**

Carefully review the result set you are getting. Are the results in the set truly different? Or are they simply being returned in the most efficient order each time. The order selected can change each time you execute the query, depending on when you last accessed the row and other factors.

**Recommendation**

If your result set must be returned in a predictable or consistent order, ensure that the SELECT statement includes an ORDER BY clause. If the result set is still returning results incorrectly, your database may be corrupt.

**See also**

- “SELECT statement [UltraLite]” on page 429
- “UltraLite database corruption” on page 462

---

**UltraLite engine client fails with error -764**

**Applies to**

Windows Mobile

**Symptom**

You are running the UltraLite engine on Windows Mobile device, and the client returns a -764 error.

**Explanation**

An error of -764 means that the engine could not be found and was unable to start.

**Recommendation**

Consider one of the following actions:

- Consider redeploying the engine to the recommended deployment location, the Windows directory. UltraLite automatically looks for the engine files in this location.

- If you have install the engine to any other location, ensure your connection code uses the START connection parameter.

- If you have used the START connection parameter, and you are sure the path to the engine is correct, ensure you have used the correct escape sequences for special characters in the path name.

For example, you may need to change this code:

```java
ULConnection conn = new ULConnection("@dbf=\Program Files\HelloEngine\HelloEngine.udb;
START=\Windows\uleng16.exe")
```

To something similar to:

```java
ULConnection conn = new ULConnection("@dbf=\\"Program Files \"\HelloEngine\HelloEngine.udb;
START=\\Windows\uleng16.exe")
```
See also

- “UltraLite engine startup” on page 117
- “UltraLite START connection parameter” on page 175
Index

Symbols

% operator
  modulo function, UltraLite, 351
&
  UltraLite bitwise operator, 274
- comment indicator
  UltraLite about, 252
--case option
  UltraLite initialize database utility (ulinit), 199
--checksum_level option
  UltraLite initialize database utility (ulinit), 199
--collation option
  UltraLite initialize database utility (ulinit), 199
--databaseid option
  UltraLite initialize database utility (ulinit), 199
--datacopy option
  UltraLite initialize database utility (ulinit), 199
--date_format option
  UltraLite initialize database utility (ulinit), 199
--date_order option
  UltraLite initialize database utility (ulinit), 199
--dba option
  UltraLite initialize database utility (ulinit), 199
--exactschema option
  UltraLite initialize database utility (ulinit), 199
--exclude option
  UltraLite initialize database utility (ulinit), 199
--fips option
  UltraLite initialize database utility (ulinit), 199
--help option
  UltraLite initialize database utility (ulinit), 199
--identity--password option
  UltraLite initialize database utility (ulinit), 199
  UltraLite load XML to database utility (ulload), 206
--identity-file option
  UltraLite initialize database utility (ulinit), 199
  UltraLite load XML to database utility (ulload), 206
--insertforupload option
  UltraLite initialize database utility (ulinit), 199
--key option
  UltraLite initialize database utility (ulinit), 199
--list_collation option
  UltraLite initialize database utility (ulinit), 199
--log option
  UltraLite initialize database utility (ulinit), 199
  UltraLite synchronization utility (ulsync), 210
--max_hash_size option
  UltraLite initialize database utility (ulinit), 199
--mirror_file option
  UltraLite initialize database utility (ulinit), 199
--no_warnings option
  UltraLite initialize database utility (ulinit), 199
--obfuscate option
  UltraLite initialize database utility (ulinit), 199
--overwrite option
  UltraLite initialize database utility (ulinit), 199
--prompt option
  UltraLite initialize database utility (ulinit), 199
--publication option
  UltraLite initialize database utility (ulinit), 199
--quiet option
  UltraLite initialize database utility (ulinit), 199
--reserve_size option
  UltraLite initialize database utility (ulinit), 199
--rootcert option
  UltraLite initialize database utility (ulinit), 199
--SAconnect option
  UltraLite initialize database utility (ulinit), 199
--scale option
  UltraLite initialize database utility (ulinit), 199
--sql option
  UltraLite initialize database utility (ulinit), 199
--sync_publication option
  UltraLite initialize database utility (ulinit), 199
--time_format option
  UltraLite initialize database utility (ulinit), 199
--timestamp_increment option
  UltraLite initialize database utility (ulinit), 199
--timestamp_with_time_zone_format option
  UltraLite initialize database utility (ulinit), 199
--utf8_encoding option
  UltraLite initialize database utility (ulinit), 199
--? option
  UltraLite initialize database utility (ulinit), 199
-a option
  ulinit creation parameters, 23
Index

UltraLite initialize database utility (ulinit), 199
UltraLite Java Edition Database Load utility (uljload), 235
UltraLite load XML to database utility (ulload), 206
-b option
   UltraLite database unload utility (ulunload), 216
   UltraLite Java Edition Database Unload utility (uljunload), 237
-c option
   UltraLite database unload utility (ulunload), 216
   UltraLite information utility (ulinfo), 198
   UltraLite initialize database utility (ulinit), 199
   UltraLite Interactive SQL utility (dbisql), 188
   UltraLite Java Edition Database Information utility (uljinfo), 234
   UltraLite Java Edition Database Load utility (uljload), 235
   UltraLite Java Edition Database Unload utility (uljunload), 237
   UltraLite load XML to database utility (ulload), 206
   UltraLite synchronization utility (ulsync), 210
   UltraLite validate database utility (ulvalid) utility, 219
-d option
   UltraLite database unload utility (ulunload), 216
   UltraLite initialize database utility (ulinit), 199
   UltraLite Interactive SQL utility (dbisql), 188
   UltraLite Java Edition Database Load utility (uljload), 235
   UltraLite Java Edition Database Unload utility (uljunload), 237
   UltraLite load XML to database utility (ulload), 206
   UltraLite SQL preprocessor utility (sqlpp), 193
-d1 option
   UltraLite Interactive SQL utility (dbisql), 188
-E option
   UltraLite load XML to database utility (ulload), 206
-e option
   UltraLite database unload utility (ulunload), 216
   UltraLite initialize database utility (ulinit), 199
   UltraLite Java Edition Database Unload utility (uljunload), 237
   UltraLite load XML to database utility (ulload), 206
   UltraLite SQL preprocessor utility (sqlpp), 193
   UltraLite validate database utility (ulvalid) utility, 219
-Ek option
   UltraLite Java Edition Database Information utility (uljinfo), 234
   UltraLite Java Edition Database Load utility (uljload), 235
   UltraLite Java Edition Database Unload utility (uljunload), 237
-f option
   UltraLite database unload utility (ulunload), 216
   UltraLite initialize database utility (ulinit), 199
   UltraLite Interactive SQL utility (dbisql), 188
   UltraLite Java Edition Database Load utility (uljload), 235
   UltraLite Java Edition Database Unload utility (uljunload), 237
   UltraLite load XML to database utility (ulload), 206
-g option
   UltraLite initialize database utility (ulinit), 199
   UltraLite load XML to database utility (ulload), 206
   UltraLite SQL preprocessor utility (sqlpp), 193
-h option
   UltraLite SQL preprocessor utility (sqlpp), 193
-i option
   UltraLite initialize database utility (ulinit), 199
   UltraLite Java Edition Database Load utility (uljload), 235
   UltraLite load XML to database utility (ulload), 206
-K option
   UltraLite initialize database utility (ulinit), 199
-k option
   UltraLite erase database utility (ulerase), 196
   UltraLite initialize database utility (ulinit), 199
   UltraLite SQL preprocessor utility (sqlpp), 193
-l option
   UltraLite database unload utility (ulunload), 216
   UltraLite initialize database utility (ulinit), 199
   UltraLite load XML to database utility (ulload), 206
-m option
   UltraLite initialize database utility (ulinit), 199
   UltraLite SQL preprocessor utility (sqlpp), 193
-n option
UltraLite database unload utility (ulunload), 216
UltraLite initialize database utility (ulinit), 199
UltraLite Java Edition Database Load utility (uljload), 235
UltraLite Java Edition Database Unload utility (uljunload), 237
UltraLite load XML to database utility (ulload), 206
UltraLite SQL preprocessor utility (sqlpp), 193

-nearest_century option

UltraLite database unload utility (ulunload), 216
UltraLite initialize database utility (ulinit), 199

-nogui option

UltraLite Interactive SQL utility (dbisql), 188

-o option

UltraLite initialize database utility (ulinit), 199
UltraLite load XML to database utility (ulload), 206
UltraLite SQL preprocessor utility (sqlpp), 193

-onerror option

UltraLite Interactive SQL utility (dbisql), 188

-p option

UltraLite erase database utility (ulerase), 196
UltraLite initialize database utility (ulinit), 199
UltraLite Java Edition Database Information utility (uljinfo), 234
UltraLite Java Edition Database Load utility (uljload), 235
UltraLite Java Edition Database Unload utility (uljunload), 237
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210

-q option

UltraLite database unload utility (ulunload), 216
UltraLite erase database utility (ulerase), 196
UltraLite information utility (ulinfo), 198
UltraLite initialize database utility (ulinit), 199
UltraLite Interactive SQL utility (dbisql), 188
UltraLite Java Edition Database Information utility (uljinfo), 234
UltraLite Java Edition Database Load utility (uljload), 235
UltraLite Java Edition Database Unload utility (uljunload), 237
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210

-ul option

UltraLite database unload utility (ulunload), 216
UltraLite Java Edition Database Information utility (uljinfo), 234
UltraLite Java Edition Database Load utility (uljload), 235
UltraLite Java Edition Database Unload utility (uljunload), 237
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210

-ud option

UltraLite database unload utility (ulunload), 216
UltraLite Java Edition Database Information utility (uljinfo), 234
UltraLite Java Edition Database Load utility (uljload), 235
UltraLite Java Edition Database Unload utility (uljunload), 237
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210

-ul option

UltraLite database unload utility (ulunload), 216
UltraLite Java Edition Database Information utility (uljinfo), 234
UltraLite Java Edition Database Load utility (uljload), 235
UltraLite Java Edition Database Unload utility (uljunload), 237
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210

-version

UltraLite Interactive SQL utility (dbisql), 188

-w option

UltraLite database unload utility (ulunload), 216
UltraLite Java Edition Database Information utility (uljinfo), 234
UltraLite Java Edition Database Load utility (uljload), 235
UltraLite Java Edition Database Unload utility (uljunload), 237
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210

-ul option

UltraLite database unload utility (ulunload), 216
UltraLite Java Edition Database Information utility (uljinfo), 234
UltraLite Java Edition Database Load utility (uljload), 235
UltraLite Java Edition Database Unload utility (uljunload), 237
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210

-s option

UltraLite database unload utility (ulunload), 216
UltraLite initialize database utility (ulinit), 199
UltraLite load XML to database utility (ulload), 206
UltraLite SQL preprocessor utility (sqlpp), 193

-SQL command

UltraLite Interactive SQL utility (dbisql), 188

-t option

UltraLite database unload utility (ulunload), 216
UltraLite initialize database utility (ulinit), 199
UltraLite Java Edition Database Unload utility (uljunload), 237
UltraLite load XML to database utility (ulload), 206
UltraLite SQL preprocessor utility (sqlpp), 193

-tw option

UltraLite database unload utility (ulunload), 216
UltraLite initialize database utility (ulinit), 199
UltraLite load XML to database utility (ulload), 206
UltraLite SQL preprocessor utility (sqlpp), 193

-ud option

UltraLite database unload utility (ulunload), 216
UltraLite initialize database utility (ulinit), 199
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210

-ud option

UltraLite database unload utility (ulunload), 216
UltraLite initialize database utility (ulinit), 199
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210

-w option

UltraLite database unload utility (ulunload), 216
UltraLite initialize database utility (ulinit), 199
UltraLite load XML to database utility (ulload), 206
UltraLite synchronization utility (ulsync), 210
UltraLite SQL preprocessor utility (sqlpp), 193

-x option
  UltraLite database unload utility (ulunload), 216
  UltraLite initialize database utility (ulinit), 199
  UltraLite Interactive SQL utility (dbisql), 188
  UltraLite SQL preprocessor utility (sqlpp), 193

-y option
  UltraLite database unload utility (ulunload), 216
  UltraLite initialize database utility (ulinit), 199
  UltraLite Java Edition Database Load utility (uljload), 235
  UltraLite Java Edition Database Unload utility (uljunload), 237
  UltraLite load XML to database utility (ulload), 206
  -Z option
    UltraLite initialize database utility (ulinit), 199
  -z option
    UltraLite initialize database utility (ulinit), 199
    UltraLite SQL preprocessor utility (sqlpp), 193

/* comment indicator
   UltraLite about, 252
// comment indicator
   UltraLite about, 252

10054
  UltraLite synchronization stream system errors, 97
130 error
  SQL code for UltraLite schema upgrade, 115
256-bit strong encryption
  UltraLite connection parameter for, 166
  UltraLite usage of, 26
?
  UltraLite input parameter, 262
@data option
  UltraLite database unload utility (ulunload), 216
  UltraLite erase database utility (ulerase), 196
  UltraLite information utility (ulinfo), 198
  UltraLite initialize database utility (ulinit), 199
  UltraLite Interactive SQL utility (dbisql), 188
  UltraLite load XML to database utility (ulload), 206
  UltraLite synchronization utility (ulsync), 210
  UltraLite validate database utility (ulvalid) utility, 219

^  UltraLite bitwise operator, 274
|  UltraLite bitwise operator, 274

~  UltraLite bitwise operator, 274

A

ABS function
  UltraLite syntax, 301
ACOS function
  UltraLite syntax, 301
ActiveSync
  deploy ActiveSync provider for UltraLite, 118
  deploying MobiLink UltraLite applications, 81
  registering applications for UltraLite clients, 120
  UltraLite deploying provider files, 118
  ActiveSync provider installation utility (mlasinst)
  registering applications for UltraLite clients, 120
  adding synchronization
    UltraLite applications, 74
  adding UltraLite indexes
    about, 55
  additional parameters
    UltraLite synchronization parameter, 83
administration tools
  UltraLite troubleshooting, 461
administration utilities
  UltraLite utilities reference, 186
AES encryption algorithm
  UltraLite deployment steps, 16
  UltraLite fips creation parameter, 140
  UltraLite fips usage, 26
  UltraLite usage, 26
aggregate expressions
  UltraLite SQL syntax, 260
aggregate functions
  UltraLite alphabetical list, 294
aliases
  UltraLite columns, 430
  UltraLite equivalents, 293
ALL search condition
  UltraLite SQL, 266
  AllowDownloadDupRows
UltraLite synchronization parameter, 83
allsync tables
   UltraLite synchronizing tables, 71
ALTER DATABASE SCHEMA FROM FILE statement
   UltraLite schema changes impact, 45
   UltraLite syntax, 397
   usage, 115
ALTER PUBLICATION statement
   UltraLite syntax, 398
ALTER SYNCHRONIZATION PROFILE statement
   UltraLite syntax, 399
ALTER TABLE statement
   UltraLite syntax, 401
ALTER USER statement
   UltraLite syntax, 404
altering
   UltraLite ALTER PUBLICATION statement, 398
   UltraLite ALTER TABLE statement, 401
   UltraLite ALTER USER statement, 404
   UltraLite columns methods, 48
altering UltraLite column definitions about, 48
ambiguous string to date conversions
   UltraLite, 142
AND
   UltraLite bitwise operators, 274
   UltraLite logical operators, 265
Android
   encryption and obfuscation, 26
ANY search condition
   UltraLite SQL, 267
application profiling
   UltraLite benchmark tests, 455
applications
   (see also UltraLite applications)
ApplyFile method
   UltraLite replacement for schema upgrade, 115
arc-cosine function
   UltraLite ACOS function, 301
arc-sine function
   UltraLite ASIN function, 303
arc-tangent function
   UltraLite ATAN function, 304
architecture diagram
   UltraLite, 1
ARGN function
   UltraLite syntax, 302
arithmetic operators
   UltraLite operators, 273
   UltraLite SQL syntax, 273
articles
   UltraLite copying method, 51
   UltraLite databases, 72
   UltraLite restrictions, 72
ASCII
   UltraLite sorting, 24
   UltraLite syntax, 303
ASCII files
   UltraLite importing, 463
ASIN function
   UltraLite syntax, 303
assembling parameters into connection strings
   UltraLite about, 31
ATAN function
   UltraLite syntax, 304
ATAN2 function
   UltraLite syntax, 305
authenticating UltraLite users about, 56
authentication
   UltraLite users, 56
authentication parameters
   UltraLite synchronization parameter, 85
authentication status
   UltraLite synchronization parameter, 86
authentication value
   UltraLite synchronization parameter, 87
autocommit
   UltraLite transaction overview, 454
AUTOINCREMENT
   UltraLite syntax, 412
   UltraLiteJ syntax, 412
average function
   UltraLite AVG function, 305
AVG function
   UltraLite syntax, 305
backing up (see backups)
backup
   UltraLite about, 42
backups
   UltraLite databases on Windows Mobile, 34
   UltraLite internal mechanism, 42
UltraLite transaction overview, 454
base 10 logarithm
   UltraLite LOG10 function, 345
benchmark testing
   UltraLite characteristics of reliable tests, 458
   UltraLite databases, 455
   UltraLite methodology overview, 457
   UltraLite SQL tests for, 457
   UltraLite test creation, 458
   UltraLite test execution and analysis, 459
   UltraLite test preparation, 458
   UltraLite types, 456
BETWEEN search condition
   UltraLite SQL, 268
BIGINT data type
   UltraLite, 277
binary
   UltraLite sorting, 24
BINARY data types
   UltraLite, 289
binary data types
   UltraLite maximum size, 8
BIT data type
   UltraLite, 278
bitwise operators
   UltraLite SQL syntax, 274
BlackBerry
   encryption and obfuscation, 26
   object store limitations, 12
uljdbtserv utility, 239
uljinfo utility, 234
uljload utility, 235
uljunload utility, 237
UltraLite Java edition system tables, 244
   utilities, 234
blob_file_base_dir database option
   UltraLite Java edition description, 229
blob_file_base_dir property
   UltraLite Java edition description, 227
browsing
   UltraLite table information, 50
   UltraLite table methods, 50
building
   UltraLite CustDB application, 123
bulk loading
   UltraLite LOAD TABLE statement, 425
BYTE_LENGTH function
   UltraLite syntax, 306
BYTE_SUBSTR function
   UltraLite syntax, 307
C
C++ APIs
   (see also UltraLite C/C++ API)
C++ applications
   (see also UltraLite C/C++)
cache
   UltraLite adjustments, 439
   UltraLite cache_allocation property, 177
cache allocation
   UltraLite configuration, 181
cache size
   UltraLite performance, 439
   UltraLite usage, 144
cache_allocation database option
   UltraLite, 181
CACHE_MAX_SIZE connection parameter
   UltraLite syntax, 158
CACHE_MIN_SIZE connection parameter
   UltraLite syntax, 159
CACHE_SIZE connection parameter
   UltraLite syntax, 160
callback
   UltraLite schema upgrade errors, 117
cascading deletes
   UltraLite limitations, 3
cascading updates
   UltraLite limitations, 3
case creation parameter
   UltraLite description, 133
CASE expression
   UltraLite NULLIF function, 356
   UltraLite SQL syntax, 259
case property
   UltraLite description, 177
case sensitivity
   LIKE search condition, 272
   UltraLite case creation parameter, 133
   UltraLite case property, 177
   UltraLite comparison operators, 264
   UltraLite strings, 252
case sensitivity considerations
   UltraLite about, 133
CAST function
   UltraLite syntax, 308
casting
  UltraLite data types list, 308
catalog
  UltraLite system tables, 220
CE_FILE connection parameter
  UltraLite syntax, 161
CEILING function
  UltraLite syntax, 310
central administration of remote databases
  UltraLite databases, 21
certificates
  UltraLite application access to encryption information, 212
CHAR data type
  UltraLite, 276
CHAR function
  UltraLite syntax, 311
CHAR_LENGTH function
  UltraLite syntax, 312
char_set property
  UltraLite description, 177
character functions
  UltraLite alphabetical list, 299
character set conversion
  passwords, 418
character sets
  UltraLite char_set property, 177
  UltraLite collation creation parameter, 135
  UltraLite databases, 24
  UltraLite on Windows, 25
  UltraLite on Windows Mobile, 25
  UltraLite strings, 252
character strings
  UltraLite embedded SQL, 192
CHARINDEX function
  UltraLite syntax, 312
CHECK constraints
  UltraLite limitations, 3
CHECK CONSTRAINTS clause
  UltraLite LOAD TABLE statement, 426
CHECKPOINT statement
  UltraLite syntax, 405
checkpointing
  UltraLite CHECKPOINT syntax, 405
checkpoints
  UltraLite performance optimization, 455
CheckpointStore
  UltraLite synchronization parameter, 83
checksum_level creation parameter
  UltraLite description, 134
checksum_level property
  UltraLite description, 177
checksums
  UltraLite checksum_level creation parameter, 134
  UltraLite checksum_level property, 177
choosing an index type
  UltraLite about, 54
client databases
  UltraLite options, 82
clients
  UltraLite MobiLink clients, 63
COALESCE function
  UltraLite syntax, 313
code points
  UltraLite, 24
collation creation parameter
  UltraLite description, 135
collation property
  UltraLite description, 177
collation sequences
  UltraLite about, 24
  UltraLite changing, 24
  UltraLite LIKE search condition, 271
collations
  UltraLite collation creation parameter, 135
  UltraLite CollationName property, 177
  UltraLite unsupported, 29
column compression
  UltraLite SQL ALTER TABLE statement, 401
column names
  UltraLite SQL syntax, 258
column names in expressions
  UltraLite about, 258
columns
  syscolumn, UltraLite Java edition system table, 245
  UltraLite adding methods, 48
  UltraLite aliases, 430
  UltraLite LOAD TABLE statement, 401
  UltraLite altering usage, 48
  UltraLite copying method, 51
  UltraLite limitations, 8
comma-separated lists
  UltraLite LIST function syntax, 343
command line utilities
  UltraLite database unload (ulunload) syntax, 216
UltraLite engine start (uleng16) syntax, 195
UltraLite engine stop (ulstop) syntax, 196
UltraLite erase database (ulerase) syntax, 196
UltraLite information (ulinfo) syntax, 197
UltraLite initialize database (ulinit) syntax, 198
UltraLite Interactive SQL (dbisql) syntax, 187
UltraLite load XML to database (ulload) syntax, 206
UltraLite SQL preprocessor (sqlpp) syntax, 192
UltraLite synchronization (ulsync) syntax, 210
UltraLite validate database (ulvalid) syntax, 219
command prompts
  Interactive SQL mode, 187
comments
  UltraLite syntax, 252
commit flush
  UltraLite configuration, 182
COMMIT statement
  UltraLite syntax, 406
COMMIT_FLUSH connection parameter
  UltraLite syntax, 163
commit_flush_count database option
  UltraLite, 182
commit_flush_count property
  UltraLite description, 177
commit_flush_timeout database option
  UltraLite, 183
commit_flush_timeout property
  UltraLite description, 177
committing
  UltraLite COMMIT syntax, 406
  UltraLite database rows, 453
  UltraLite transaction overview, 454
comparing
  UltraLite and SQL Anywhere databases, 3
comparing UltraLite and SQL Anywhere about, 3
comparison operators
  UltraLite dynamic SQL syntax, 264
  UltraLite SQL, 264
comparisons
  UltraLite search conditions, 263
compatibility
  UltraLite SQL, 264
compressed columns
  UltraLite ALTER TABLE statement, 401
computed columns
  UltraLite limitations, 3
  COMPUTES clause
    UltraLite LOAD TABLE statement, 426
CON connection parameter
  UltraLite syntax, 164
concatenating strings
  UltraLite string operators, 273
concurrency
  UltraLite issues, 452
  UltraLite synchronization, 452
conditions
  UltraLite ALL conditions, 266
  UltraLite ANY, 267
  UltraLite BETWEEN, 268
  UltraLite EXISTS, 268
  UltraLite IN, 269
  UltraLite search, 263
connCount property
  UltraLite description, 177
connecting
  MobiLink UltraLite Stream Type synchronization parameter, 98
  UltraLite database troubleshooting, 31
  connecting to an UltraLite database about, 31
connection methods
  UltraLite about, 31
connection parameters
  alphabetical list (UltraLite), 156
  CE, 169
  DBN for UltraLite, 167
  iPhone, 169
  Mac, 168
  NT, 168
  UltraLite, 31
  UltraLite CACHE_MAX_SIZE, 158
  UltraLite CACHE_MIN_SIZE, 159
  UltraLite CACHE_SIZE, 160
  UltraLite CE_FILE, 161
  UltraLite choosing between, 33
  UltraLite COMMIT_FLUSH, 163
  UltraLite CON, 164
  UltraLite connection summary, 31
  UltraLite DBF, 165
  UltraLite DBKEY, 166
  UltraLite desktop, 168
  UltraLite device, 169
  UltraLite file_name, 165
  UltraLite key, 166
creating reference databases for UltraLite, 29
UltraLite CREATE PUBLICATION statement, 409
UltraLite CREATE TABLE statement, 411
UltraLite databases, 19
UltraLite databases at a command prompt, 20
UltraLite databases from a MobiLink sync model, 20
UltraLite databases from XML, 21
UltraLite databases using central administration of remote databases, 21
UltraLite databases with ulinit, 198
UltraLite indexes, 55
UltraLite publications, 72
UltraLite publications tables, 78
UltraLite remote databases, 64
UltraLite table methods, 47
UltraLite users with connection parameters, 58
UltraLite users with SQL statements, 60
UltraLite users with Sybase Central, 61
creating databases
   Sybase Central UltraLite plug-in, 19
   UltraLite creation parameters, 131
   UltraLite remote database, 64
creating publications
   UltraLite databases with MobiLink applications, 72
creating remote databases
   UltraLite clients, 64
creating UltraLite databases
   about, 19
creating UltraLite tables
   about, 47
creation parameters (UltraLite)
   about, 131
case, 133
checksum_level, 134
collation, 135
date_format, 136
date_order, 138
fips creation parameter, 140
max_hash_size, 141
nearest_century, 142
obfuscate, 143
page_size, 144
precision, 146
scale, 147
time_format, 148
timestamp_format, 150
timestamp_increment, 152
utf8_encoding, 155
CROSS JOIN clause
   UltraLite syntax, 422
CSV files
   UltraLite importing, 463
CURRENT DATE function
   UltraLite TODAY function, 388
CURRENT DATE special value
   UltraLite syntax, 253
current row
   UltraLite concurrency, 452
CURRENT TIME special value
   UltraLite syntax, 254
CURRENT_TIMESTAMP special value
   UltraLite syntax, 255
cursors
   UltraLite current row, 452
   UltraLite dirty reads, 40
CustDB
   UltraLite about, 12
   UltraLite application readme files, 14
   UltraLite sample file locations, 15
custdb.db
   location of, 15
custdb.sql
   calling synchronization scripts, 123
custdb.udb
   UltraLite location of, 15
cycles
   UltraLite foreign key issues, 73
   UltraLite foreign keys, 73
D
daemon
   running the UltraLite engine as a, 195
data
   UltraLite loading, 206
   UltraLite selecting rows, 429
   UltraLite unloading, 216
   UltraLite viewing methods, 50
UltraLiteJ selecting rows, 429
data consistency
  UltraLite dirty reads, 40
data management
  UltraLite components, 16
  UltraLite description, 1
  UltraLite state information, 452
Data Manager
  UltraLite database storage, 33
data sources
  UltraLite CustDB tutorial example, 123
data type conversion functions
  UltraLite about, 295
data type conversions
  UltraLite CAST, 308
data types
  spatial (UltraLite), 292
  UltraLite about, 275
  UltraLite alias equivalents, 293
  UltraLite casting of values, 308
  UltraLite limitations, 8
  UltraLite special value, 253
  UltraLite SQL conversion functions, 295
data types in UltraLite
  about, 275
database files
  (see also UltraLite databases)
  UltraLite connection parameters, 33
  UltraLite encrypting, 166
  UltraLite maximum size, 8
database management
  UltraLite layers of, 1
database objects
  UltraLite copying method, 51
database options
  (see also database options (UltraLite))
  UltraLite, 181
  UltraLite Java edition, 228
database options (UltraLite)
  browsing, 35
  cache_allocation, 181
  commit_flush_count, 182
  commit_flush_timeout, 183
  DB_PROPERTY function, 328
  global_database_id, 184
  isolation_level, 185
  ml_remote_id, 185
  SET OPTION statement, 431
database options window
  accessing, 35
database page size considerations
  UltraLite about, 144
database properties
  (see also database properties (UltraLite))
  UltraLite, 177
  UltraLite Java edition, 227
database properties (UltraLite)
  browsing, 35
  creation parameters, 23
  fips usage, 26
  obfuscate usage, 26
  utf8_encoding usage, 24
database schemas
  UltraLite system tables, 220
database sizes
  UltraLite limit, 8
database utilities
  UltraLite database connections, 31
database validation
  UltraLite, 41
database_name property
  UltraLite Java edition description, 227
databases
  (see also UltraLite databases)
  comparing UltraLite and SQL Anywhere, 3
  creating with UltraLite plug-in, 19
  UltraLite loading bulk data into, 425
DATALENGTH function
  UltraLite syntax, 319
date considerations
  UltraLite about, 136
DATE data type
  UltraLite, 284
DATE function
  UltraLite syntax, 320
date functions
  UltraLite alphabetical list, 295
date parts
  about, 295
  available in UltraLite, 136
date_format creation parameter
  UltraLite description, 136
date_format database option
  UltraLite Java edition description, 229
date_format property
  UltraLite description, 177
UltraLite Java edition description, 227

date_order creation parameter
UltraLite description, 138
date_order database option
UltraLite Java edition description, 230
date_order property
UltraLite description, 177
UltraLite Java edition description, 227

DATEADD function
UltraLite syntax, 321

DATEDIFF function
UltraLite syntax, 322

DATEFORMAT function
UltraLite syntax, 323

DATENAME function
UltraLite syntax, 324

DATEPART function
UltraLite syntax, 325

dates
query the current system date, 333
UltraLite ambiguous string conversions, 142
UltraLite conversion functions, 295
UltraLite date_format property, 177
UltraLite date_order property, 177
UltraLite formatting, 136
UltraLite Java edition date_format property, 227
UltraLite Java edition date_order property, 227
UltraLite ordering, 138
UltraLite rollover point, 142
datetime
UltraLite conversion functions, 295

DATETIME data type
UltraLite, 285

DATETIME function
UltraLite syntax, 325

DAY function
UltraLite syntax, 326
day of week
UltraLite DOW function, 330

DAYNAME function
UltraLite syntax, 326

DAYS function
UltraLite syntax, 327

DB_PROPERTY function
UltraLite syntax, 328

DBF connection parameter
UltraLite syntax, 165
dbqsql utility

UltraLite exit codes, 192
UltraLite syntax, 187
UltraLite troubleshooting data imports, 463
dbqsql.com
about, 191
dbqsql.exe
about, 191
shutting down before installing, 461
DBKEY connection parameter
UltraLite syntax, 166
dblgen16.dll
ActiveSync conduit deployment in UltraLite, 118

DBN connection parameter
UltraLite syntax, 167

DDL
UltraLite schema changes with, 45
deadlocks
UltraLite prevention of, 454

DECIMAL data type
UltraLite, 279
decimal point position considerations
UltraLite precision, 146
UltraLite scale creation parameter, 147
decimal precision
UltraLite precision, 146
decimals
UltraLite, 253
declaring default global autoincrement columns
UltraLite clients in MobiLink systems, 67

DEFAULT TIMESTAMP columns
UltraLite syntax, 412
UltraLiteJ syntax, 412
default values
UltraLite CURRENT DATE, 253
UltraLite CURRENT TIME, 254
UltraLite CURRENT TIMESTAMP, 255
UltraLite CURRENT UTC TIMESTAMP, 255
UltraLite SQLCODE, 256
defaults
UltraLite autoincrement, 412
UltraLiteJ autoincrement, 412

DEFAULTS clause
UltraLite LOAD TABLE statement, 426

DEGREES function
syntax, 329
delaying commits
performance enhancements, 455

DELETE statement
UltraLite dynamic SQL syntax, 418
UltraLiteJ dynamic SQL syntax, 418
deleting
UltraLite columns, 401
UltraLite databases, 453
UltraLite indexes, 56
UltraLite publications, 80
UltraLite START SYNCHRONIZATION
DELETE statement, 432
UltraLite table methods, 49
UltraLite TRUNCATE TABLE statement, 435
UltraLite users with connection parameters, 58
UltraLite users with SQL statements, 60
UltraLite users with Sybase Central, 62
UltraLiteJ START SYMBASE CENTRAL
DELETE statement, 432
UltraLiteJ TRUNCATE TABLE statement, 435
deleting data
UltraLite file size impact of, 463
DELIMITED BY clause
UltraLite LOAD TABLE statement, 426
demos
(see also tutorials)
deploying
applications that use ActiveSync for UltraLite clients, 81
changes to UltraLite database files, 107
UltraLite ActiveSync provider files, 118
UltraLite databases, 115
UltraLite engine troubleshooting, 465
UltraLite FIPS-enabled applications, 140
UltraLite schema upgrades, 115
UltraLite to devices, 107
UltraLite upgrades to devices, 107
derived tables
UltraLite FROM clause, 422
UltraLite SQL, 261
Designing synchronization in UltraLite
about, 68
desktop connection parameter
UltraLite syntax, 168
desktop creation
UltraLite about, 19
device connection parameter
UltraLite syntax, 169
devices
deploying UltraLite to, 107
UltraLite deployment techniques, 107
UltraLite multiple connection parameters for, 33
diagnostics
UltraLite benchmark tests, 455
DIFFERENCE function
UltraLite syntax, 330
digits
UltraLite maximum number, 146
direct page scans
UltraLite about, 448
dirty reads
UltraLite isolation levels, 40
disable concurrency
UltraLite synchronization parameter overview, 452
DisableConcurrency
UltraLite synchronization parameter, 83
DISTINCT keyword
UltraLite SQL, 430
distinct operation
UltraLite execution plans, 450
DOUBLE data type
UltraLite, 280
double hyphen
UltraLite comment indicator, 252
double slash
UltraLite comment indicator, 252
DOW function
UltraLite syntax, 330
download acknowledgements
UltraLite send_download_ack synchronization parameter, 96
download only
UltraLite synchronization parameter, 88
download only synchronization
UltraLite download_only synchronization parameter, 88
download only tables
UltraLite non-synchronizing tables, 70
download-only synchronization
UltraLite defining overview, 74
UltraLite synchronization parameter, 88
download_only synchronization parameter
UltraLite reference, 88
DROP INDEX statement
UltraLite syntax, 419
UltraLiteJ syntax, 419
DROP PUBLICATION statement
UltraLite syntax, 420
DROP SYNCHRONIZATION PROFILE statement
UltraLite syntax, 420
UltraLiteJ syntax, 420

DROP TABLE statement
UltraLite syntax, 421
UltraLiteJ syntax, 421

DROP USER statement
UltraLite syntax, 422

dropping
UltraLite columns, 401
UltraLite DROP SYNCHRONIZATION PROFILE statement, 420
UltraLite DROP USER statement, 422
UltraLite indexes, 56
UltraLite SQL CREATE INDEX statement, 407
UltraLite SQL DROP INDEX statement, 419
UltraLite SQL DROP PUBLICATION statement, 420
UltraLite SQL DROP TABLE statement, 421
UltraLite table methods, 49
UltraLiteJ DROP SYNCHRONIZATION PROFILE statement, 420
UltraLiteJ SQL DROP INDEX statement, 419
UltraLiteJ SQL DROP TABLE statement, 421

dropping an index
UltraLite about, 56
dropping publications
UltraLite clients, 80
dropping UltraLite tables about, 49
dummy operation
UltraLite execution plans, 450
dynamic SQL
UltraLite arithmetic operators, 273
UltraLite bitwise operators, 274
UltraLite logical operators, 265
UltraLite operator precedence, 274
UltraLite string operators, 273

embedded SQL library functions (see UltraLite embedded SQL library functions)
empty databases
UltraLite populating after running ulinit, 205
encoding
UltraLite utf8_encoding creation parameter, 155
UltraLite utf8_encoding usage, 24
ENCODING clause
UltraLite LOAD TABLE statement, 426
encryption
UltraLite changing key, 140
UltraLite data deployment considerations, 16
UltraLite deployment steps, 16
UltraLite encryption keys, 166
UltraLite encryption property, 177
UltraLite fips creation parameter, 140
UltraLite fips property usage, 26
UltraLite fips usage, 26
UltraLite obfuscate creation parameter, 143
UltraLite obfuscate property usage, 26
UltraLite synchronization deployment considerations, 107
UltraLite TLS synchronization configuration, 107
UltraLiteJ development, 26
encryption keys
UltraLite changing, 140
encryption property
UltraLite description, 177
END
UltraLite CASE expression, 259
ENDIF
UltraLite IF expressions, 259
engines
(see also UltraLite engine)
entity-relationship diagrams
UltraLite about, 51
entity-relationship tab
UltraLite using, 51
environment variables
ERRORLEVEL for UltraLite, 192
UltraLite ULSQLCONNECT, 32
UltraLite usage, 31
ER diagram tab
UltraLite about, 51
erase database utility
UltraLite syntax, 196
error callback
UltraLite schema upgrade errors, 117
error codes
UltraLite database synchronization utility (ulsync), 187
UltraLite load XML to database utility (ulload), 187
UltraLite unload data to XML utility (unload), 187
UltraLite utilities, 187
ERRORLEVEL environment variable
Interactive SQL return code for UltraLite, 192
errors
UltraLite client error -764, 465
UltraLite SQLE_DATABASE_ERROR, 462
UltraLite SQLE_DEVICE_ERROR, 462
UltraLite SQLE_MEMORY_ERROR, 462
escape sequences
UltraLite engine paths, 465
ESCAPES clause
UltraLite LOAD TABLE statement, 427
ESQL (see embedded SQL) (see UltraLite SQL)
event notifications
UltraLite working with, 36
examples
(see also samples)
(see also tutorials)
exclusive OR
UltraLite bitwise operator, 274
execution plans
UltraLite checking for index usage, 446
UltraLite how to read, 450
UltraLite operations, 450
UltraLite overriding, 446
UltraLite text of, 449
UltraLite viewing, 449
UltraLite working with, 446
EXISTS search condition
UltraLite search conditions, 263
UltraLite SQL, 268
exit codes
Interactive SQL utility (dbisql) for UltraLite, 192
UltraLite database synchronization utility (ulsync), 187
UltraLite load XML to database utility (ulload), 187
UltraLite unload data to XML utility (unload), 187
UltraLite CustDB sample location, 15
Excel operation
UltraLite syntax, 331
EXPLANATION function
UltraLite syntax, 331
exponential function
UltraLite EXP function, 331
exponents
UltraLite, 253
export tools
UltraLite unload utility, 216
exporting
UltraLite databases with unload, 216
expressions
UltraLite aggregate, 260
UltraLite CASE expressions, 259
UltraLite column names, 258
UltraLite constants, 258
UltraLite IF expressions, 259
UltraLite input parameters, 262
UltraLite SQL, 256
UltraLite SQL operator precedence, 274
UltraLite subqueries, 261
expressions in UltraLite
about, 256
F
failures
UltraLite preventing out-of-memory errors, 174
UltraLite schema upgrade errors, 117
features
UltraLite comparison list, 3
Federal Information Processing Standards (see FIPS)
fetches
UltraLite, 40
fetching rows
UltraLite concurrency, 40
file names
UltraLite connection parameters, 33
file property
UltraLite description, 177
file size
UltraLite database troubleshooting, 463
file systems
(see also VFS)
files
transferring with MLFileTransfer, 76
UltraLite ActiveSync provider, 118
UltraLite CustDB sample location, 15
filter operation
Index

UltraLite execution plans, 450
FIPS
  UltraLite encrypted database deployment, 16
  UltraLite fips creation parameter, 140
  UltraLite fips property usage, 26
  UltraLite setup and deployment, 140
  UltraLite TLS-enabled synchronization, 107
FIPS creation parameter
  UltraLite description, 140
fips database property
  UltraLite deployment steps, 16
  UltraLite usage, 26
fips network protocol option
  UltraLite TLS-enabled synchronization, 107
FIRST clause
  UltraLite SELECT statement, 429
  UltraLiteJ SELECT statement, 429
FLOAT data type
  UltraLite, 280
FLOOR function
  UltraLite syntax, 332
flush count
  UltraLite formatting, 182
flush timeout
  UltraLite formatting, 183
flushing
  UltraLite databases, 182
footprint
  UltraLite databases, 3
FOR clause
  UltraLite SELECT statement, 430
FOR READ ONLY clause
  UltraLite direct page scans with, 448
FORCE ORDER clause
  UltraLite SELECT statement, 431
foreign key columns
  UltraLite Java edition system table for, 246
foreign key cycles
  UltraLite about, 73
  UltraLite issues, 73
foreign keys
  UltraLite characteristics, 53
  UltraLite copying method, 51
  UltraLite foreign keys, 412
  UltraLite Java edition system table for, 246
  UltraLite of unnamed, 412
  UltraLiteJ foreign keys, 412
  UltraLiteJ of unnamed, 412
FORMAT clause
  UltraLite LOAD TABLE statement, 427
format options (UltraLite)
  utf8_encoding usage, 24
FROM clause
  UltraLite LOAD TABLE statement, 426
  UltraLite SELECT statement, 430
  UltraLite syntax, 422
functions
  return NULL if you specify NULL argument, 294
types of function for UltraLite, 294
  UltraLite aggregate, 294
  UltraLite data type conversion SQL, 295
  UltraLite date and time, 295
  UltraLite introduction, 294
  UltraLite miscellaneous, 297
  UltraLite numeric, 298
  UltraLite string, 299
  UltraLite system SQL, 300
functions, aggregate
  about, 294
  UltraLite AVG, 305
  UltraLite COUNT, 317
  UltraLite LIST, 343
  UltraLite MAX, 347
  UltraLite MIN, 347
  UltraLite SUM, 384
functions, data type conversion
  UltraLite about, 295
  UltraLite CAST, 308
  UltraLite CONVERT, 314
  UltraLite HEXTOINT, 334
  UltraLite INTTOHEX, 338
  UltraLite ISDATE, 339
  UltraLite ISNULL, 339
functions, date and time
  SWITCHOFFSET, 385
  TODATETIMEOFFSET, 387
  UltraLite about, 295
  UltraLite DATE, 320
  UltraLite DATEADD, 321
  UltraLite DATEDIFF, 322
  UltraLite DATEFORMAT, 323
  UltraLite DATENAME, 324
  UltraLite DATEPART, 325
  UltraLite DATETIME, 325
  UltraLite DAY, 326
  UltraLite DAYNAME, 326
Index

SYNC_PROFILE_OPTION_VALUE, 386
UltraLite DATALENGTH, 319
UltraLite DB_PROPERTY, 328
fundamentals
UltraLite database management, 452

G
GETDATE function
query the current system date, 333
UltraLite syntax, 333
GetLastIdentity method
UltraLite synchronization, 66
getScriptVersion method
UltraLite example, 104
getStream method
UltraLite example, 98
getUploadOK method
UltraLite example, 101
global autoincrement
exhausted range in UltraLite, 65
UltraLite clients in MobiLink systems, 64
UltraLite global_database_id, 184
UltraLite, setting, 65
UltraLite, setting defaults, 67
global database ID considerations
UltraLite about, 184
global database identifier
UltraLite global_database_id, 184
UltraLite, setting, 65
global_database_id option
UltraLite, 184
UltraLite CREATE TABLE statement, 412
UltraLite, setting, 65
UltraLiteJ CREATE TABLE statement, 412
global_database_id property
UltraLite description, 177
UltraLite Java edition description, 227
globally unique identifiers
UltraLite clients in MobiLink systems, 64
UltraLite SQL syntax for NEWID function, 354
graphical plans
UltraLite not supported, 449
GREATER function
UltraLite syntax, 334
GROUP BY clause
UltraLite SELECT statement, 430
group-by operation

UltraLite execution plans, 450
GUIDs
UltraLite clients in MobiLink systems, 64
UltraLite SQL syntax for NEWID function, 354
UltraLite SQL syntax for STRTOUUID function, 382
UltraLite SQL syntax for UUIDTOSTR function, 391

H
hash
UltraLite configuring size for, 445
UltraLite max_hash_size property, 177
UltraLite optimal size for, 443
UltraLite size considerations, 141
hashing
UltraLite indexes, 141
HAVING clause
UltraLite search conditions, 263
hexadecimal strings
UltraLite about, 334
HEXTOINT function
UltraLite syntax, 334
host
UltraLite ULsynchronize arguments, 99
HOUR function
UltraLite syntax, 335
HOURS function
UltraLite syntax, 336

I
IDENTIFIED BY password clause
ALTER USER statement, 405
identifiers
UltraLite SQL, 251
IDs
ml_remote_id option, 185
UltraLite global database, 184
IF expressions
UltraLite SQL syntax, 259
IF NOT EXISTS clause
CREATE PUBLICATION statement [UltraLite], 409
DROP INDEX statement [UltraLite] [UltraLiteJ], 419
DROP PUBLICATION statement [UltraLite] [UltraLiteJ], 419
DROP SYNCHRONIZATION PROFILE statement [UltraLite] [UltraLiteJ], 419
DROP TABLE statement [UltraLite] [UltraLiteJ], 419
UltraLite, CREATE TEXT INDEX statement, 407
IFNULL function
UltraLite syntax, 337
ignored rows
UltraLite synchronization parameter, 89
ignored_rows synchronization parameter
UltraLite reference, 89
importing data
UltraLite troubleshooting, 463
importing data into databases
UltraLite unload utility, 206
IN search condition
UltraLite SQL, 269
in-process runtime
UltraLite about, 16
index performance considerations
UltraLite about, 141
index scans
UltraLite, 441
index-based UltraLite optimizations
UltraLite about, 441
index-scan operation
UltraLite execution plans, 450
indexes
sysindex, UltraLite Java edition system table, 247
sysindexcolumn, UltraLite Java edition system table, 248
UltraLite bypassing use of, 448
UltraLite copying method, 51
UltraLite creating, 55
UltraLite deleting, 56
UltraLite determining which index is scanned, 446
UltraLite hash considerations, 141
UltraLite hash value, 141
UltraLite introduction of, 441
UltraLite limitations, 8
UltraLite non-unique index characteristics, 54
UltraLite page_size usage, 144
UltraLite performance enhancements, 441
UltraLite primary keys, 29
UltraLite sysindex system table, 222
UltraLite sysixcol system table, 223
UltraLite types, 54
UltraLite unique index characteristics, 54
UltraLite unique key characteristics, 54
UltraLite UNIQUE SQL parameter, 407
UltraLite when to create, 55
UltraLite when to use, 54
UltraLite working with, 52
indicators
UltraLite comments in SQL block, 252
initializing
UltraLite databases with ulinit, 198
initializing databases
Sybase Central UltraLite plug-in, 19
INNER JOIN clause
UltraLite syntax, 422
inner references
UltraLite subqueries, 261
input parameters
UltraLite about, 262
INPUT statement
UltraLite troubleshooting, 463
INSERT statement
UltraLite Interactive SQL example, 51
UltraLite syntax, 424
UltraLiteJ syntax, 424
inserting
UltraLite data using LOAD TABLE statement, 425
UltraLite INSERT statement, 424
UltraLite rows in bulk, 425
UltraLiteJ INSERT statement, 424
INSERTSTR function
UltraLite syntax, 338
installation
UltraLite troubleshooting, 461
installing
UltraLite on devices, 107
INTEGER data type
UltraLite, 281
integrity
UltraLite CREATE TABLE statement, 412
UltraLiteJ CREATE TABLE statement, 412
integrity constraints
UltraLite usage, 412
UltraLiteJ usage, 412
Interactive SQL
command line, 191
UltraLite command line, 187
UltraLite displaying plans, 446
UltraLite plan interpretation, 450
UltraLite plan operations, 450
UltraLite text plans, 449
UltraLite troubleshooting data imports, 463
Interactive SQL utility (dbisql)
  UltraLite exit codes, 192
  UltraLite syntax, 187
INTTOHEX function
  UltraLite syntax, 338
iPad (see iOS)
iPhone (see iOS)
IS
  UltraLite logical operators, 265
ISDATE function
  UltraLite syntax, 339
ISNULL function
  UltraLite syntax, 339
isolation levels
  UltraLite about, 39
  UltraLite dirty reads, 40
  UltraLite isolation levels, 39
isolation_level option
  UltraLite, 185

J
join operation
  UltraLite execution plans, 450
joins
  UltraLite FROM clause syntax, 422

K
keep partial download synchronization parameter
  UltraLite reference , 89
key connection parameter
  UltraLite syntax, 166
KEY JOIN clause
  UltraLite syntax, 422
keys
  UltraLite index creation from, 54
  UltraLite index hash, 141
  UltraLite primary, 47
keyset operation
  UltraLite execution plans, 450
keywords
  UltraLite SQL, 251

L
LCASE function
  UltraLite syntax, 340
LEFT function
  UltraLite syntax, 340
LEFT OUTER JOIN clause
  UltraLite syntax, 422
LENGTH function
  UltraLite syntax, 341
LESSER function
  UltraLite syntax, 342
libraries
  UltraLite choices, 16
  UltraLite deploying uleng to Windows Mobile, 16
  UltraLite FIPS-enabled applications, 140
libulbase.lib
  UltraLite C++ development, 107
libulrsa.lib
  UltraLite C++ development, 107
libulrt.lib
  UltraLite C++ development, 107
LIKE search condition
  UltraLite case-sensitivity, 271
  UltraLite collations, 271
  UltraLite pattern length, 270
  UltraLite search conditions, 263
  UltraLite SQL, 270
like-scan operation
  UltraLite execution plans, 450
limitations
  UltraLite, 8
  UltraLite data types, 275
limits
  UltraLite, 8
line length
  UltraLite sqlpp utility output, 192
linking
  UltraLite runtime libraries, 16
Linux
  UltraLite runtime libraries for, 107
LIST function
  UltraLite syntax, 343
lists
  UltraLite LIST function syntax, 343
literals
  UltraLite constants, 258
LOAD TABLE statement
  UltraLite syntax, 425
loading
  UltraLite bulk inserts, 425
  UltraLite LOAD TABLE statement, 425
loading data
  UltraLite LOAD TABLE statement, 425
loading databases
  UltraLite databases with ulload, 206
LOCATE function
  UltraLite syntax, 343
locking
  UltraLite concurrency, 454
LOG function
  UltraLite syntax, 344
LOG10 function
  UltraLite syntax, 345
logging
  UltraLite internal mechanism, 42
logic
  UltraLite capturing for synchronization design, 68
logical operators
  UltraLite SQL syntax, 265
lojoin operation
  UltraLite execution plans, 450
LONG BINARY data type
  UltraLite, 290
LONG VARCHAR data type
  UltraLite, 276
LOWER function
  UltraLite syntax, 345
lowercase strings
  UltraLite LCASE function, 340
  UltraLite LOWER function, 345
LTRIM function
  UltraLite syntax, 346

M

maintaining
  UltraLite on devices, 107
maintaining primary key uniqueness
  UltraLite clients in MobiLink systems, 64
management tools
  UltraLite utilities reference, 186
managing databases
  UltraLite data and state, 452
managing temporary tables
  UltraLite about, 447
managing transactions
  UltraLite increased throughput, 455
mathematical expressions
  UltraLite arithmetic operators, 273
MAX function
  UltraLite syntax, 347
max_hash_size creation parameter
  UltraLite description, 141
max_hash_size property
  UltraLite description, 177
maximum
  UltraLite date ranges, 275
  UltraLite physical limits, 8
maximum length
  UltraLite user ID, 176
mechanism
  UltraLite application and database deployment, 107
media failures
  UltraLite transaction overview, 454
memory
  UltraLite limits, 8
memory failures
  UltraLite preventing, 174
memory usage
  UltraLite database storage, 33
  UltraLite indexes, 441
  UltraLite row states, 453
  UltraLite SQL performance, 457
metadata
  UltraLite considering for reserve size, 174
MIN function
  UltraLite syntax, 347
minimum
  UltraLite date ranges, 275
MINUTE function
  UltraLite syntax, 348
MINUTES function
  UltraLite syntax, 348
MIRROR_FILE connection parameter
  UltraLite syntax, 170
ml_add_connection_script system procedure
  adding, 123
ml_add_table_script system procedure
  adding, 123
ML_GET_SERVER_NOTIFICATION syntax, 350
ml_remote_id option
  UltraLite, 185
  UltraLite property configuration, 197
ml_remote_id property
  UltraLite description, 177
UltraLite Java edition description, 227
mlasdesk.dll
deploying UltraLite applications, 118
mlasdev.dll
deploying UltraLite applications, 118
mlasinst utility
registering applications for UltraLite clients, 120
UltraLite deploying with DLL files, 118
MobiLink
UltraLite clients, 63
UltraLite CREATE PUBLICATION statement, 409
UltraLite SQL statements, 395
UltraLite user ID uniqueness, 185
MobiLink ActiveSync provider installation utility (mlasinst)
registering applications for UltraLite clients, 120
MobiLink clients
UltraLite progress counter, 2
MobiLink file transfers
UltraLite client overview of, 76
MobiLink project
UltraLite sample, 123
MobiLink synchronization
setting timestamp_increment creation parameter, 153
UltraLite clients, 63
MOD function
UltraLite syntax, 351
modeling
UltraLite databases from MobiLink, 20
modifying
UltraLite columns, 401
UltraLite tables, 401
MONEY data type
UltraLite equivalent, 293
monitoring synchronization
UltraLite observer synchronization parameter, 91
UltraLite setObserver method, 91
MONTH function
UltraLite syntax, 352
MONTHNAME function
UltraLite syntax, 352
MONTHS function
UltraLite syntax, 353
multi-table joins
UltraLite databases, 3
multi-threaded
UltraLite applications, 452
multiple databases
UltraLite maximum of, 452
multiple devices
UltraLite connection parameters for, 33
N
name property
UltraLite description, 177
names
UltraLite column names, 258
NATURAL JOIN clause
UltraLite syntax, 422
nearest century conversion considerations
UltraLite about, 142
nearest_century creation parameter
UltraLite description, 142
nearest_century database option
UltraLite Java edition description, 230
nearest_century property
UltraLite description, 177
UltraLite Java edition description, 227
network protocols
UltraLite supported, 3
UltraLite Sync Result synchronization parameter, 100
UltraLite synchronization using HTTP, 98
UltraLite synchronization using HTTPS, 98
UltraLite synchronization using TCP/IP, 98
new password
UltraLite synchronization parameter, 90
NEWID function
UltraLite syntax, 354
newmobilinkpwd synchronization parameter
UltraLite reference, 90
Non-repeatable reads
UltraLite about, 40
non-unique indexes
UltraLite characteristics, 53
UltraLite index creation from, 54
nosync tables
UltraLite non-synchronizing tables, 69
NOT
UltraLite bitwise operator, 274
UltraLite logical operators, 265
NOW function
UltraLite syntax, 355
NULL
 returned by functions if a NULL argument is specified, 294
 UltraLite ISNULL function, 339
 NULL values
 UltraLite SQL, 253
 NULLIF function
 UltraLite about, 356
 UltraLite using with CASE expressions, 260
 number of authentication parameters
 UltraLite synchronization parameter, 91
 numbers
 UltraLite SQL, 253
 NUMERIC data type
 UltraLite, 282
 numeric functions
 UltraLite alphabetical list, 298
 numeric precision
 UltraLite precision, 146

O
 obfuscate database creation parameter
 UltraLite description, 143
 obfuscate database property
 UltraLite usage, 26
 obfuscation
 UltraLite usage, 26
 UltraLiteJ development, 26
 observer synchronization parameter
 UltraLite description, 91
 ON phrase
 UltraLite search conditions, 263
 on-device creation
 about, 22
 OpenSSL
 UltraLite cryptographic module, 26
 UltraLite TLS-enabled synchronization, 107
 operator precedence
 UltraLite SQL syntax, 274
 operators
 UltraLite arithmetic operators, 273
 UltraLite bitwise operators, 274
 UltraLite comparison operators, 264
 UltraLite logical operators, 265
 UltraLite precedence of operators, 274
 UltraLite SQL syntax, 272
 UltraLite string operator, 273
 optimization
 UltraLite benchmark tests, 455
 UltraLite checkpoints, 455
 UltraLite execution plan access options, 446
 UltraLite indexes, 443
 UltraLite queries, 430
 UltraLite SQL, 446
 optimizer
 (see also query optimizer)
 UltraLite execution plan access options, 446
 UltraLite impact of, 446
 UltraLite overriding, 446
 UltraLite plan interpretation, 450
 UltraLite plan operations, 450
 UltraLite using, 446
 optimizing UltraLite
 query performance, 439
 options
 UltraLite browsing, 35
 UltraLite database unload utility (ulunload), 216
 UltraLite erase database utility (ulerase), 196
 UltraLite information utility (ulinfo), 198
 UltraLite initialize database utility (ulinit), 199
 UltraLite Interactive SQL utility (dbisql), 188
 UltraLite Java edition, 228
 UltraLite load XML to database utility (ulload), 206
 UltraLite SQL preprocessor utility (sqlpp), 193
 UltraLite synchronization profiles, 213
 UltraLite synchronization utility (ulsync), 210
 UltraLite validate database utility (ulvalid) utility, 219
 options (UltraLite)
 cache_allocation, 181
 commit_flush_count, 182
 commit_flush_timeout, 183
 DBPROPERTY function, 328
 global_database_id, 184
 isolation_level, 185
 ml_remote_id, 185
 SET OPTION statement, 431
 OR
 UltraLite bitwise operators, 274
 UltraLite logical operators, 265
 Oracle consolidated databases
 UltraLite issues with, 63
ORDER BY clause
  UltraLite SELECT statement, 430
  UltraLite troubleshooting with, 464
order of operations
  UltraLite SQL operator precedence, 274
ordering result sets
  UltraLite troubleshooting, 464
outer references
  UltraLite subqueries, 261
overflow errors
  AVG function, 306
  SUM function, 385
overhead
  UltraLite considering for reserve size, 174

P
packed rows
  UltraLite about, 46
packing rows
  UltraLite effect of, 145
page sizes
  UltraLite Java edition page_size property, 227
  UltraLite page_size property, 177
page_size creation parameter
  UltraLite description, 144
page_size property
  UltraLite description, 177
  UltraLite Java edition description, 227
pages
  UltraLite size considerations, 144
parameters
  UltraLite connection list, 156
  UltraLite connection overview, 31
  UltraLite database creation, 131
  UltraLite SQL input, 262
partial download retained synchronization parameter
  UltraLite reference, 92
partition sizes
  UltraLite defaults, choosing, 65
  UltraLite defaults, overriding, 67
  UltraLite exhausting defaults, 65
partitioning
  UltraLite primary keys, 65
  UltraLite rows publishing, 79
password
  UltraLite synchronization parameter, 92
Password connection parameter
  UltraLite syntax, 172
passwords
  character set conversion, 418
  maximum length, 418
  PWD UltraLite connection parameter, 172
  salt value for UltraLite connection parameter, 173
  UltraLite databases, 56
  UltraLite new MobiLink password parameter, 90
  UltraLite password synchronization parameter, 92
paths
  UltraLite connection parameters, 33
  UltraLite engine, 461
PATINDEX function
  UltraLite syntax, 356
pattern length
  UltraLite LIKE search condition, 270
pattern matching
  UltraLite case-sensitivity, 271
  UltraLite collations, 271
  UltraLite LIKE search condition, 270
  UltraLite PATINDEX function, 356
  UltraLite pattern length, 270
performance
  separating commits from checkpoints, 455
  UltraLite CACHE_MAX_SIZE connection parameter, 158
  UltraLite CACHE_MIN_SIZE connection parameter, 159
  UltraLite CACHE_SIZE connection parameter, 160
  UltraLite choosing optimal hash size, 443
  UltraLite download-only synchronization, 88
  UltraLite index hashing, 141
  UltraLite page sizes, 144
  UltraLite preventing memory failures, 174
  UltraLite query optimization, 430
  UltraLite query optimization techniques, 439
  UltraLite query tuning with index hashing, 441
  UltraLite specifying FOR READ ONLY clause, 446
  UltraLite upload only synchronization, 101
  UltraLite using index, 54
  UltraLite using index for applications, 29
  UltraLite using index for improving query performance, 407
  UltraLite using index for large tables, 52
  UltraLite viewing an execution plan, 448
persistent memory
UltraLite database storage, 33
phantom rows
  UltraLite, 40
physical limitations
  UltraLite, 8
PI function
  UltraLite syntax, 359
ping
  UltraLite synchronization parameter, 93
placeholder
  UltraLite SQL input parameter, 262
planning
  UltraLite synchronization design overview, 68
plans
  UltraLite cursors, 331
  UltraLite operations for, 450
  UltraLite plan interpretation, 450
  UltraLite plan operations, 450
  UltraLite queries, overriding, 446
  UltraLite queries, reading, 450
  UltraLite queries, working with, 446
  UltraLite syntax, 331
  UltraLite text plans, 449
platforms
  UltraLite file storage, 33
  UltraLite multiple connection parameters for, 33
plug-ins
  UltraLite troubleshooting, 461
pools
  UltraLite unused Global IDs, 65
populating
  UltraLite databases created with ulinit, 205
port number
  UltraLite ULSSynchronize arguments, 99
POWER function
  UltraLite syntax, 359
precedence
  UltraLite SQL operator precedence, 274
precision creation parameter
  UltraLite description, 146
precision database option
  UltraLite Java edition description, 231
precision property
  UltraLite description, 177
  UltraLite Java edition description, 227
predicates
  UltraLite ALL, 266
  UltraLite ANY, 267
UltraLite BETWEEN, 268
UltraLite comparison operators, 264
UltraLite EXISTS, 268
UltraLite IN, 269
UltraLite LIKE search condition, 270
UltraLite SQL, 263
prefix
  UltraLite for Windows Mobile databases, 34
prepared statements
  UltraLite input parameters, 262
primary key indexes
  UltraLite bypassing use of, 448
primary keys
  UltraLite characteristics, 53
  UltraLite generating unique values, 354
  UltraLite generating unique values using UUIDs, 354
  UltraLite indexing, 29
  UltraLite integrity constraints, 412
  UltraLite order of columns, 412
  UltraLite order of columns, 73
  UltraLite tables, 47
  UltraLite troubleshooting data imports, 463
  UltraLite UUIDs and GUIDs, 354
  UltraLiteJ integrity constraints, 412
  UltraLiteJ order of columns, 412
procedures
  UltraLite limitations, 3
  UltraLite offset mismatches with, 2
properties
  DB_PROPERTY function, 328
  UltraLite browsing, 35
  UltraLite database creation parameters, 23
  UltraLite databases, 177
  UltraLite Java edition databases, 227
  UltraLite Java edition system tables, 249
properties (UltraLite)
  DB_PROPERTY function, 328
provider files
  UltraLite deploying ActiveSync, 118
providers
  UltraLite ActiveSync files, 118
public certificate
  UltraLite application access to, 212
publication creation wizard
  UltraLite creating publications, 72
publications
sysarticles, UltraLite Java edition system table, 244
syspublications, UltraLite Java edition system table, 248
UltraLite ALTER PUBLICATION statement, 398
UltraLite copying method, 51
UltraLite CREATE PUBLICATION statement, 409
UltraLite design plans with, 68
UltraLite dropping, 80
UltraLite Java edition database schema description for, 248
UltraLite Java edition table listing in schema, 244
UltraLite limit, 198
UltraLite publication synchronization parameter, 94
UltraLite publishing overview, 72
UltraLite publishing tables, 78
UltraLite rows publishing, 79
UltraLite SQL CREATE INDEX statement, 407
UltraLite SQL DROP INDEX statement, 419
UltraLite SQL DROP PUBLICATION statement, 420
UltraLite SQL DROP TABLE statement, 421
UltraLite SQL DROP TABLE statement, 421
UltraLite synchronization, 94
UltraLite synchronization parameter for, 94
UltraLite sysarticle system table, 221
UltraLite syspublication system table, 224
UltraLite WHERE clause usage, 79
UltraLite working with, 77
UltraLiteJ SQL DROP INDEX statement, 419
UltraLiteJ SQL DROP TABLE statement, 421
publishing
UltraLite rows, 79
UltraLite tables, 78
UltraLite whole table, 72
publishing whole tables
UltraLite, 78
PWD connection parameter
UltraLite syntax, 172
query optimization
(see also optimizer)
UltraLite SQL, 446
query optimizer
(see also optimizer)
UltraLite, 446
QUOTES clause
UltraLite LOAD TABLE statement, 427
R
RADIANS function
UltraLite syntax, 360
RAND function
syntax, 361
UltraLiteJ syntax, 361
random numbers
RAND function, 361
UltraLiteJ RAND, 361
range
UltraLite date type, 275
read committed
UltraLite isolation levels, 39
read uncommitted
UltraLite isolation levels, 39
read-only tables
UltraLite databases, 74
UltraLite synchronizing, 74
reading
UltraLite table rows, 40
reading UltraLite access plans about, 450
readme files
UltraLite CustDB applications, 14
REAL data type
UltraLite, 282
recovery
UltraLite, 42
UltraLite about, 42
UltraLite introduction, 453
UltraLite transaction overview, 454
recovery from system failure
UltraLite internal mechanism, 42
reference databases
about, 29
creating for UltraLite, 29
options for UltraLite, 29
referential integrity
UltraLite databases, 3
UltraLite indexes, 54
UltraLite table order, 73
registering
  MobiLink UltraLite applications with ActiveSync, 120
reload.sql
  UltraLite loading, 425
REMAINDER function
  UltraLite syntax, 362
remote databases
  creating UltraLite clients, 64
  UltraLite synchronization count, 2
remote IDs
  setting in UltraLite databases, 185
remote servers
  UltraLite CREATE TABLE statement, 411
removing
  UltraLite users with connection parameters, 58
  UltraLite users with SQL statements, 60
  UltraLite users with Sybase Central, 62
removing data
  UltraLite file size impact of, 463
renaming
  UltraLite database objects during upgrade, 115
  UltraLite tables, 401
REPEAT function
  UltraLite syntax, 363
REPLACE function
  UltraLite syntax, 363
REPLICATE function
  UltraLite syntax, 364
requests
  UltraLite concurrency, 452
  UltraLite management of, 452
RESERVE_SIZE connection parameter
  UltraLite syntax, 174
reserved words
  UltraLite SQL, 251
restartable downloads
  UltraLite keep partial download, 89
  UltraLite partial download retained, 92
  UltraLite resume partial download, 95
restoring
  UltraLite transaction overview, 454
result sets
  UltraLite troubleshooting unpredictable changes, 464
resume partial download synchronization parameter
  UltraLite reference, 95
return codes
  Interactive SQL utility (dbisql) for UltraLite, 192
RIGHT function
  UltraLite syntax, 365
RIGHT OUTER JOIN clause
  UltraLite syntax, 422
role names
  UltraLite foreign keys, 412
  UltraLite role names, 412
  UltraLiteJ foreign keys, 412
  UltraLiteJ role names, 412
ROLLBACK statement
  UltraLite syntax, 429
  UltraLiteJ syntax, 429
rollbacks
  UltraLite databases, 453
rolling back
  UltraLite transaction overview, 454
  UltraLite transactions, 429
  UltraLiteJ transactions, 429
ROUND function
  UltraLite syntax, 366
rounding
  UltraLite about, 46
  UltraLite effect of, 145
  UltraLite observing, 174
row limit operation
  UltraLite execution plans, 450
rows
  UltraLite deleting all rows from a table, 435
  UltraLite fetching, 40
  UltraLite INSERT statement, 424
  UltraLite inserting in bulk, 425
  UltraLite locking of, 454
  UltraLite publishing, 79
  UltraLiteJ deleting all rows from a table, 435
  UltraLiteJ INSERT statement, 424
RSA encryption algorithm
  UltraLite TLS-enabled synchronization, 107
RTRIM function
  UltraLite syntax, 366
running
  UltraLite CustDB application, 123
runtime libraries
    UltraLite, 16
runtime runtimes
    (see also UltraLite runtime)

S
salt value for password
    UltraLite syntax, 173
sample applications
    CustDB in UltraLite, 12
samples
    (see also examples)
    (see also tutorials)
SAP HANA
    Interactive SQL utility (dbisql), 187
scale creation parameter
    UltraLite description, 147
scale database option
    UltraLite Java edition description, 231
scale property
    UltraLite description, 177
    UltraLite Java edition description, 227
scan operation
    UltraLite execution plans, 450
scanning
    UltraLite database pages in queries, 448
    UltraLite indexes in queries, 441
schema
    UltraLite catalog of tables for, 45
    UltraLite changes, precautions, 45
schemas
    UltraLite SQL ALTER DATABASE SCHEMA
    FROM FILE syntax, 397
    UltraLite system tables, 220
    working with UltraLite, 45
scjview
    terminating before installing, 461
script versions
    UltraLite getScriptVersion, 104
    UltraLite setScriptVersion method, 104
    UltraLite version synchronization parameter, 104
scripted upload
    UltraLite CREATE PUBLICATION syntax, 409
search conditions
    UltraLite ALL, 266
    UltraLite ANY, 267
    UltraLite BETWEEN, 268
    UltraLite EXISTS, 268
    UltraLite IN, 269
    UltraLite LIKE, 270
    UltraLite SQL, 263
SECOND function
    UltraLite syntax, 367
SECONDS function
    UltraLite syntax, 368
security
    UltraLite, 26
    UltraLite AES_FIPS database deployment, 16
    UltraLite FIPS encryption, 140
    UltraLite overview of, 26
    UltraLite TLS-enabled synchronization, 107
SELECT statement
    UltraLite copying result sets with, 51
    UltraLite syntax, 429
    UltraLite troubleshooting result sets, 464
    UltraLiteJ syntax, 429
    UltraLiteJ SELECT statement, 429
    UltraLiteJ SELECT statement, 429
send download acknowledgement
    UltraLite synchronization parameter, 96
    send_download_ack synchronization parameter
    UltraLite reference, 96
sensitivity
    UltraLite case creation parameter, 133
serialized transaction
    UltraLite processing of, 454
SET OPTION statement
    UltraLite syntax, 431
setObserver method
    UltraLite example, 91
setScriptVersion method
    UltraLite example, 104
setStream method
    UltraLite example, 98
setStreamParms method
    UltraLite example, 99
setting the global database identifier
    UltraLite clients in MobiLink systems, 65
setting the hash size
    about, 445
    UltraLite example, 102
setUserData method
    UltraLite syntax, 369
SHORT_PLAN function
SIGN function
  UltraLite syntax, 370
SIMILAR function
  UltraLite syntax, 370
SIN function
  UltraLite syntax, 371
SKIP clause
  UltraLite LOAD TABLE statement, 427
slash-asterisk
  UltraLite comment indicator, 252
SMALLINT data type
  UltraLite, 283
SMALLMONEY data type
  UltraLite equivalent, 293
sort order
  UltraLite collations, 25
SOUNDEX function
  UltraLite syntax, 372
SPACE function
  UltraLite syntax, 372
spaces
  UltraLite filepath definitions, 461
spatial
  list of supported functions (UltraLite), 292
  spatial data
    about (UltraLite), 291
    introduction (UltraLite), 291
spatial data types
  ST_GEOMETRY (UltraLite), 292
spatial functions
  ST_AsBinary, UltraLite, 373
  ST_AsExtText, UltraLite, 374
  ST_AsText, UltraLite, 374
  ST_Distance, UltraLite, 375
  ST_Equals, UltraLite, 375
  ST_IntersectsRect, UltraLite, 376
  ST_Point, UltraLite, 376
  ST_PointFromExtText, UltraLite, 377
  ST_PointFromText UltraLite, 378
  ST_PointFromWKB, UltraLite, 378
  ST_SRID, UltraLite, 379
  ST_X, UltraLite, 379
  ST_Y, UltraLite, 380
Spatial SQL API
  ST_Geometry type for UltraLite, 292
special values
  UltraLite CURRENT DATE, 253
  UltraLite CURRENT TIME, 254
  UltraLite CURRENT TIMESTAMP, 255
  UltraLite CURRENT UTC TIMESTAMP, 255
  UltraLite SQL, 253
  UltraLite SQLCODE, 256
SQL
  (see also UltraLite SQL)
    data types in UltraLite, 275
    UltraLite comparison operators, 264
    UltraLite expressions, 256
    UltraLite identifiers, 251
    UltraLite keywords, 251
    UltraLite numbers, 253
    UltraLite operators, 272
    UltraLite reserved words, 251
    UltraLite schema changes with, 45
    UltraLite search conditions, 263
    UltraLite statement types, 396
    UltraLite strings, 252
    UltraLite variables, 275
SQL Anywhere databases
  database comparison with UltraLite, 3
SQL code
  UltraLite schema upgrade errors, 117
SQL Flagger
  UltraLite usage, 192
SQL functions
  return NULL if you specify NULL argument, 294
  types of function for UltraLite, 294
  UltraLite aggregate, 294
  UltraLite data type conversion, 295
  UltraLite date and time, 295
  UltraLite introduction, 294
  UltraLite miscellaneous, 297
  UltraLite numeric, 298
  UltraLite string, 299
  UltraLite system, 300
SQL preprocessor utility (sqlpp)
  UltraLite syntax , 192
SQL statements
  alphabetical list of UltraLite statements, 395
  UltraLite, 395
SQL syntax
  UltraLite CASE expression, 259
  UltraLite column names, 258
  UltraLite comments, 252
  UltraLite constants, 258
  UltraLite functions, 294
  UltraLite IF expressions, 259
Index

UltraLite input parameters, 262
UltraLite predicates, 263
UltraLite special values, 253
UltraLite SQLCODE special value, 256
SQLCODE
  UltraLite concurrency checks, 454
  UltraLite special value, 256
SQLCODE SQLE_LOCKED
  UltraLite concurrency error, 454
SQLE_CONVERSION_ERROR
  UltraLite upgrade warning, 117
SQLE_DATABASE_ERROR
  UltraLite data corruption, 462
SQLE_DEVICE_ERROR
  UltraLite data corruption, 462
SQLE_DOWNLOAD_CONFLICT error
  UltraLite synchronization, 74
SQLE_MAX_ROW_SIZE_EXCEEDED
  UltraLite error, 46
SQLE_MEMORY_ERROR
  UltraLite data corruption, 462
SQLE_NOTFOUND
  UltraLite concurrency error, 454
SQLE_ROW_DROPPED_DURING_SCHEMA_UPGRADE
  UltraLite upgrade warning, 115
SQLE_UNABLE_TO_CONNECT_OR_START
  troubleshooting, 461
sqlpp utility
  UltraLite syntax, 192
SQRT function
  UltraLite syntax, 373
square root function
  UltraLite SQRT function, 373
ST_AsBinary
  function, UltraLite, 373
ST_AsExtText
  function, UltraLite, 374
ST_AsText
  function, UltraLite, 374
ST_Distance
  function, UltraLite, 375
ST_Equals
  function, UltraLite, 375
ST_GEOMETRY
  type for UltraLite, 292
ST_IntersectsRect
  function, UltraLite, 376
ST_Point
  function, UltraLite, 376
ST_PointFromExtText
  function, UltraLite, 377
ST_PointFromText
  function, UltraLite, 378
ST_PointFromWKB
  function, UltraLite, 378
ST_SRID
  function, UltraLite, 379
ST_X
  function, UltraLite, 379
ST_Y
  function, UltraLite, 380
START connection parameter
  UltraLite for uleng16 on Windows Mobile, 16
  UltraLite syntax, 175
START SYNCHRONIZATION DELETE statement
  UltraLite syntax, 432
  UltraLiteJ syntax, 432
state bytes
  UltraLite databases, 453
state management
  UltraLite overview, 452
statements
  UltraLite, 395
  UltraLite prepared, input parameters for, 262
  UltraLite statement performance tuning, 457
  UltraLite types, 396
STOP SYNCHRONIZATION DELETE statement
  UltraLite syntax, 433
  UltraLiteJ syntax, 433
storage
  UltraLite limits, 8
  UltraLite reserve size, 174
stored procedures
  UltraLite limitations, 3
STR function
  UltraLite syntax, 380
stream error
  UltraLite synchronization parameter, 97
stream parameters
  UltraLite synchronization parameter, 99
stream synchronization parameters
  UltraLite synchronization parameter, 99
stream type
  UltraLite synchronization parameter, 98
stream_error synchronization parameter
<table>
<thead>
<tr>
<th>Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>UltraLite reference</td>
<td>97</td>
</tr>
<tr>
<td>UltraLite ul_stream_error structure</td>
<td>97</td>
</tr>
<tr>
<td>string_parms synchronization parameter</td>
<td>99</td>
</tr>
<tr>
<td>STRING function</td>
<td></td>
</tr>
<tr>
<td>UltraLite syntax</td>
<td>381</td>
</tr>
<tr>
<td>string functions</td>
<td></td>
</tr>
<tr>
<td>UltraLite alphabetical list</td>
<td>299</td>
</tr>
<tr>
<td>string length</td>
<td></td>
</tr>
<tr>
<td>UltraLite LENGTH function</td>
<td>341</td>
</tr>
<tr>
<td>string literals</td>
<td></td>
</tr>
<tr>
<td>UltraLite constants</td>
<td>258</td>
</tr>
<tr>
<td>string operators</td>
<td></td>
</tr>
<tr>
<td>UltraLite dynamic SQL syntax</td>
<td>273</td>
</tr>
<tr>
<td>string position</td>
<td></td>
</tr>
<tr>
<td>UltraLite LOCATION function</td>
<td>343</td>
</tr>
<tr>
<td>strings</td>
<td></td>
</tr>
<tr>
<td>UltraLite case sensitivity</td>
<td>252</td>
</tr>
<tr>
<td>UltraLite maximum size</td>
<td>8</td>
</tr>
<tr>
<td>UltraLite nearest_century conversions to dates with</td>
<td>142</td>
</tr>
<tr>
<td>UltraLite removing trailing blanks</td>
<td>366</td>
</tr>
<tr>
<td>UltraLite replacing</td>
<td>363</td>
</tr>
<tr>
<td>UltraLite SQL</td>
<td>252</td>
</tr>
<tr>
<td>UltraLite SQL functions</td>
<td>299</td>
</tr>
<tr>
<td>STRIP clause</td>
<td></td>
</tr>
<tr>
<td>UltraLite LOAD TABLE statement</td>
<td>427</td>
</tr>
<tr>
<td>strong encryption</td>
<td></td>
</tr>
<tr>
<td>UltraLite deployment steps</td>
<td>16</td>
</tr>
<tr>
<td>UltraLite fips creation parameter</td>
<td>140</td>
</tr>
<tr>
<td>UltraLite usage</td>
<td>26</td>
</tr>
<tr>
<td>STRTOUUID function</td>
<td></td>
</tr>
<tr>
<td>UltraLite syntax</td>
<td>382</td>
</tr>
<tr>
<td>STUFF function</td>
<td></td>
</tr>
<tr>
<td>UltraLite syntax</td>
<td>382</td>
</tr>
<tr>
<td>subqueries</td>
<td></td>
</tr>
<tr>
<td>UltraLite SQL</td>
<td>261</td>
</tr>
<tr>
<td>subquery operation</td>
<td></td>
</tr>
<tr>
<td>UltraLite execution plans</td>
<td>450</td>
</tr>
<tr>
<td>SUBSCRIBE BY clause</td>
<td></td>
</tr>
<tr>
<td>UltraLite synchronization limitations</td>
<td>72</td>
</tr>
<tr>
<td>SUBSTR function</td>
<td></td>
</tr>
<tr>
<td>UltraLite syntax</td>
<td>383</td>
</tr>
<tr>
<td>SUBSTRING function</td>
<td></td>
</tr>
<tr>
<td>UltraLite syntax</td>
<td>383</td>
</tr>
<tr>
<td>substrings</td>
<td></td>
</tr>
<tr>
<td>UltraLite about</td>
<td>383</td>
</tr>
<tr>
<td>UltraLite replacing</td>
<td>363</td>
</tr>
<tr>
<td>SUM function</td>
<td></td>
</tr>
<tr>
<td>UltraLite syntax</td>
<td>384</td>
</tr>
<tr>
<td>supplying UltraLite connection parameters</td>
<td>about</td>
</tr>
<tr>
<td>switches</td>
<td></td>
</tr>
<tr>
<td>UltraLite database unload utility (ulunload)</td>
<td>216</td>
</tr>
<tr>
<td>UltraLite erase database utility (ulerase)</td>
<td>196</td>
</tr>
<tr>
<td>UltraLite information utility (ulinfo)</td>
<td>198</td>
</tr>
<tr>
<td>UltraLite initialize database utility (ulinit)</td>
<td>199</td>
</tr>
<tr>
<td>UltraLite Interactive SQL utility (dbisql)</td>
<td>188</td>
</tr>
<tr>
<td>UltraLite load XML to database utility (ulload)</td>
<td>206</td>
</tr>
<tr>
<td>UltraLite SQL preprocessor utility (sqlpp)</td>
<td>193</td>
</tr>
<tr>
<td>UltraLite synchronization utility (ulsync)</td>
<td>210</td>
</tr>
<tr>
<td>UltraLite validate database utility (ulvalid) utility</td>
<td>219</td>
</tr>
<tr>
<td>SWITCHOFFSET function</td>
<td></td>
</tr>
<tr>
<td>syntax</td>
<td>385</td>
</tr>
<tr>
<td>Sybase Central</td>
<td></td>
</tr>
<tr>
<td>troubleshooting UltraLite connections</td>
<td>461</td>
</tr>
<tr>
<td>sync result</td>
<td></td>
</tr>
<tr>
<td>UltraLite synchronization parameter</td>
<td>100</td>
</tr>
<tr>
<td>SYNC_PROFILE_OPTION_VALUE function</td>
<td>syntax</td>
</tr>
<tr>
<td>synchronization</td>
<td></td>
</tr>
<tr>
<td>character sets in UltraLite</td>
<td>24</td>
</tr>
<tr>
<td>setting timestamp_increment creation parameter</td>
<td>153</td>
</tr>
<tr>
<td>ulsync utility for UltraLite databases</td>
<td>210</td>
</tr>
<tr>
<td>UltraLite ALTER TABLE statement</td>
<td>401</td>
</tr>
<tr>
<td>UltraLite application implementation</td>
<td>74</td>
</tr>
<tr>
<td>UltraLite client-specific data</td>
<td>71</td>
</tr>
<tr>
<td>UltraLite clients</td>
<td>63</td>
</tr>
<tr>
<td>UltraLite CREATE TABLE statement</td>
<td>412</td>
</tr>
<tr>
<td>UltraLite CustDB tutorial</td>
<td>123</td>
</tr>
<tr>
<td>UltraLite design overview</td>
<td>68</td>
</tr>
<tr>
<td>UltraLite download_only parameter</td>
<td>88</td>
</tr>
<tr>
<td>UltraLite excluding tables with publications</td>
<td>72</td>
</tr>
<tr>
<td>UltraLite foreign keys</td>
<td>73</td>
</tr>
<tr>
<td>UltraLite ignored rows</td>
<td>89</td>
</tr>
<tr>
<td>UltraLite introduction</td>
<td>63</td>
</tr>
<tr>
<td>UltraLite monitoring</td>
<td>91</td>
</tr>
<tr>
<td>UltraLite progress counting mechanism</td>
<td>2</td>
</tr>
<tr>
<td>UltraLite read-only tables</td>
<td>74</td>
</tr>
<tr>
<td>UltraLite referential integrity</td>
<td>73</td>
</tr>
<tr>
<td>UltraLite schema changes during</td>
<td>45</td>
</tr>
<tr>
<td>UltraLite SQLE_DOWNLOAD_CONFLICT error</td>
<td>74</td>
</tr>
</tbody>
</table>
UltraLite stopping, 91
UltraLite syssyncresult system table, 224
UltraLite system table, 226
UltraLite task overview, 63
UltraLite upload only parameter, 101
UltraLiteJ CREATE TABLE statement, 412
synchronization logic
browsing Sybase Central in UltraLite, 123
synchronization models
UltraLite databases, 20
synchronization parameters
UltraLite, 82
UltraLite additionalparms, 83
UltraLite Authentication Value, 87
UltraLite Disable Concurrency overview, 452
UltraLite download_only, 88
UltraLite getScriptVersion, 104
UltraLite getStream method, 98
UltraLite getUploadOK method, 101
UltraLite keep partial download, 89
UltraLite newmobilinkpwd, 90
UltraLite observer, 91
UltraLite partial download retained, 92
UltraLite password, 92
UltraLite ping, 93
UltraLite publication, 94
UltraLite required, 82
UltraLite resume partial download, 95
UltraLite send_download_ack, 96
UltraLite setObserver method, 91
UltraLite setScriptVersion method, 104
UltraLite setStream method, 98
UltraLite setStreamParms method, 99
UltraLite setUserData method, 102
UltraLite stream type, 98
UltraLite stream_error, 97
UltraLite stream_parms, 99
UltraLite Sync Result, 100
UltraLite upload_ok, 101
UltraLite upload_only, 101
UltraLite user_data, 102
UltraLite user_name, 103
UltraLite version, 104
synchronization profile options
about, UltraLite clients, 213
synchronization profiles
UltraLite ALTER SYNCHRONIZATION PROFILE statement, 399
UltraLite CREATE SYNCHRONIZATION PROFILE statement, 410
UltraLite DROP SYNCHRONIZATION PROFILE statement, 420
UltraLite SYNCHRONIZE statement, 434
UltraLiteJ DROP SYNCHRONIZATION PROFILE statement, 420
UltraLiteJ SYNCHRONIZE statement, 434
synchronization scripts
browsing the UltraLite sample, 123
synchronization stream parameters
UltraLite stream type, 98
synchronization streams
UltraLite getStream method, 98
UltraLite setStream method, 98
UltraLite setStreamParms method, 99
UltraLite setting, 98
UltraLite stream synchronization parameter, 98
UltraLite stream_error synchronization parameter, 97
UltraLite stream_parms synchronization parameter, 99
UltraLite ULHTTPSStream, 98
UltraLite ULHTTPStream, 98
SYNCHRONIZE statement
UltraLite syntax, 434
UltraLiteJ syntax, 434
synchronizing
UltraLite databases on Windows Mobile, 81
synchronizing data
UltraLite file size impact of, 463
syntax
UltraLite arithmetic operators, 273
UltraLite bitwise operators, 274
UltraLite CASE expression, 259
UltraLite column names, 258
UltraLite comparison operators, 264
UltraLite constants, 258
UltraLite CURRENT DATE special value, 253
UltraLite CURRENT TIMESTAMP special value, 255
UltraLite CURRENT UTC TIMESTAMP special value, 255
UltraLite IF expressions, 259
UltraLite logical operators, 265
UltraLite predicates, 263
UltraLite search conditions, 263
UltraLite special values, 253
UltraLite SQL comments, 252
UltraLite SQL CURRENT TIME special value, 254
UltraLite SQL functions, 294
UltraLite SQL input parameters, 262
UltraLite SQL operator precedence, 274
UltraLite SQL operators, 272
UltraLite SQLCODE special value, 256
UltraLite string operators, 273

SYS
  UltraLite system tables, 220
  sysarticle system table [UltraLite]
    about, 221
  sysarticles system table [UltraLite Java edition databases]
    about, 244
  syscolumn system table [UltraLite Java edition databases]
    about, 245
  syscolumn system table [UltraLite]
    about, 221
  sysfkcol system table [UltraLite Java edition databases]
    about, 246
  sysforeignkey system table [UltraLite Java edition databases]
    about, 246
  sysindex system table [UltraLite Java edition databases]
    about, 247
  sysindex system table [UltraLite]
    about, 222
  sysindexcolumn system table [UltraLite Java edition databases]
    about, 248
  sysixcol system table [UltraLite]
    about, 223
  syssyncresult system table [UltraLite]
    about, 224
  syssyncresult system table [UltraLite Java edition databases]
    about, 248
  syssyncresult system table [UltraLite]
    about, 224
  systable system table [UltraLite Java edition databases]
    about, 249
  systable system table [UltraLite]
    about, 226

system failure
  UltraLite recover from, 42
system failures
  UltraLite transaction overview , 454
system functions
  UltraLite about, 300
  UltraLite limitations, 3
system tables
  UltraLite about, 220
  UltraLite browsing methods, 50
  UltraLite Java edition sysarticles, 244
  UltraLite Java edition syscolumn, 245
  UltraLite Java edition sysfkcol, 246
  UltraLite Java edition sysforeignkey, 246
  UltraLite Java edition sysindex, 247
  UltraLite Java edition sysindexcolumn, 248
  UltraLite Java edition syspublications, 248
  UltraLite Java edition systable, 249
  UltraLite Java edition sysuldata, 249
  UltraLite sysarticle, 221
  UltraLite syscolumn, 221
  UltraLite sysindex, 222
  UltraLite sysixcol, 223
  UltraLite syspublication, 224
  UltraLite syssyncresult, 224
  UltraLite systable, 226

system_error_code values
  UltraLite synchronization stream errors, 97
sysuldata system table [UltraLite Java edition databases]
  about, 249

T
  table constraints
    UltraLite adding, deleting, or modifying, 401
    UltraLite CREATE TABLE statement, 412
    UltraLiteJ CREATE TABLE statement, 412
  table expressions
    UltraLite subqueries, 261
  table owners
    UltraLite, 251
  table size
    number of rows, 8
    UltraLite limit, 8
  TableOrder
    UltraLite synchronization parameter, 83
Index

tables
  UltraLite allsync, 71
  UltraLite ALTER TABLE statement, 401
  UltraLite browsing methods, 50
  UltraLite bulk loading, 425
  UltraLite controlling synchronization with publications, 72
  UltraLite copying method, 51
  UltraLite copying methods, 50
  UltraLite CREATE TABLE statement, 411
  UltraLite creating methods, 47
  UltraLite deleting methods, 49
  UltraLite download only, 70
  UltraLite editing methods, 50
  UltraLite INSERT statement, 424
  UltraLite limitations, 8
  UltraLite nosync, 69
  UltraLite order of, 73
  UltraLite publications, 72
  UltraLite rows publishing, 79
  UltraLite size limit, 46
  UltraLite temporary table usage, 448
  UltraLite TRUNCATE TABLE statement, 435
  UltraLite working with, 46
  UltraLiteJ CREATE TABLE statement, 411
  UltraLiteJ INSERT statement, 424
  UltraLiteJ TRUNCATE TABLE statement, 435
TAN function
  UltraLite syntax, 386
TCP/IP
  (see also TCP/IP synchronization)
temp operation
  UltraLite execution plans, 450
TEMP_DIR connection parameter
  UltraLite syntax, 175
temporary files
  UltraLite limitation, 175
temporary tables
  UltraLite about, 447
  UltraLite limitations, 8
  UltraLite queries, 448
  UltraLite synchronization using, 71
terminating processes
  UltraLite troubleshooting upgrades, 461
TEXT data type
  UltraLite equivalent, 293
text plans
  UltraLite viewing, 449
  THEN
    UltraLite IF expressions, 259
threads
  UltraLite concurrency, 452
time considerations
  UltraLite about, 148
TIME data type
  UltraLite, 285
time functions
  UltraLite alphabetical list, 295
time_format creation parameter
  UltraLite description, 148
time_format database option
  UltraLite Java edition description, 232
time_format property
  UltraLite description, 177
  UltraLite Java edition description, 227
times
  UltraLite conversion functions, 295
TIMESTAMP data type
  UltraLite, 286
  UltraLite column limitations, 3
TIMESTAMP special value
  UltraLite column limitations, 3
  UltraLite TIMESTAMP columns, 412
  UltraLiteJ TIMESTAMP columns, 412
TIMESTAMP WITH TIME ZONE data type
  UltraLite, 287
timestamp_format creation parameter
  UltraLite description, 150
timestamp_format database option
  UltraLite Java edition description, 232
timestamp_format property
  UltraLite description, 177
  UltraLite Java edition description, 227
timestamp_increment creation parameter
  UltraLite description, 152
  using in MobiLink synchronization, 153
timestamp_increment database option
  UltraLite Java edition description, 233
timestamp_increment property
  UltraLite description, 177
  UltraLite Java edition description, 227
timestamp_with_time_zone property
  UltraLite description, 177
  UltraLite Java edition description, 227
timestamp_with_time_zone_format creation parameter
UltraLite, 153
timestamp_with_time_zone_format database option
    UltraLite Java edition description, 233
timestamps
    UltraLite timestamp_format creation parameter, 150
    UltraLite timestamp_increment creation parameter, 152
TINYINT data type
    UltraLite, 283
TODATETIMEOFFSET function
    syntax, 387
TODAY function
    UltraLite syntax, 388
TOP clause
    UltraLite SELECT statement, 429
    UltraLiteJ SELECT statement, 429
tracking
    UltraLite synchronization, 2
transaction logging
    UltraLite internal mechanism, 42
transaction management
    UltraLite, 452
transaction processing
    checkpoints and commits, 455
    UltraLite COMMIT_FLUSH connection parameter, 163
    UltraLite commit_flush_count option, 182
    UltraLite commit_flush_timeout option, 183
transactions
    UltraLite checkpoints for, 405
    UltraLite COMMIT statement, 406
    UltraLite concurrency, 452
    UltraLite databases, 453
    UltraLite flushing, 182
    UltraLite isolation level, 40
    UltraLite rolling back, 429
    UltraLite schema changes impact, 45
    UltraLiteJ rolling back, 429
transferring files
    UltraLite files with MLFileTransfer, 76
transport-layer security
    (see also TLS)
triggers
    UltraLite limitations, 3
TRIM function
    UltraLite syntax, 388
troubleshooting
    avoiding synchronization issues with foreign key cycles, 73
    UltraLite backing up application, 63
    UltraLite checksum failures, 134
    UltraLite connection parameter precedence, 31
    UltraLite getUploadOK method, 101
    UltraLite global ID numbers, 65
    UltraLite implementing resumable downloads, 74
    UltraLite maintaining timestamps and timestamp increments, 150
    UltraLite ping synchronization parameter, 93
    UltraLite retrieving GLOBAL AUTOINCREMENT value, 66
    UltraLite Stream Error synchronization parameter, 97
    UltraLite Sync Result synchronization parameter, 100
    UltraLite upload_ok synchronization parameter, 101
TRUNCATE function
    UltraLite syntax, 389
TRUNCATE TABLE statement
    UltraLite syntax, 435
    UltraLiteJ syntax, 435
truncating
    UltraLite tables, 435
    UltraLiteJ tables, 435
TRUNCNUM function
    UltraLite syntax, 389
trusted certificates
    UltraLite application access to encryption information, 212
tuning performance
    UltraLite max_hash_size, 141
    UltraLite with index hashing, 441
tutorials
    UltraLite CustDB, 123
UCASE function
    UltraLite syntax, 389
UDB
    (see also UltraLite databases)
UID connection parameter
    UltraLite syntax, 176
ul_stream_error structure
    UltraLite example, 97
<table>
<thead>
<tr>
<th>UL_SYNC_ALL macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>UltraLite publications list, 94</td>
</tr>
<tr>
<td>(see also UltraLite APIs)</td>
</tr>
<tr>
<td>(see also UltraLite applications)</td>
</tr>
<tr>
<td>(see also UltraLite databases)</td>
</tr>
<tr>
<td>(see also UltraLite embedded SQL)</td>
</tr>
<tr>
<td>(see also UltraLite SQL)</td>
</tr>
<tr>
<td>(see also UltraLite utilities)</td>
</tr>
<tr>
<td>about, 1</td>
</tr>
<tr>
<td>architecture, 1</td>
</tr>
<tr>
<td>collations unsupported, 29</td>
</tr>
<tr>
<td>concurrency, 452</td>
</tr>
<tr>
<td>CustDB sample, 123</td>
</tr>
<tr>
<td>data conversion, 295</td>
</tr>
<tr>
<td>database creation parameters, 131</td>
</tr>
<tr>
<td>deploying applications and databases, 107</td>
</tr>
<tr>
<td>deploying databases, 115</td>
</tr>
<tr>
<td>engine and runtime for, 16</td>
</tr>
<tr>
<td>engine starting, 195</td>
</tr>
<tr>
<td>engine stopping, 196</td>
</tr>
<tr>
<td>environment variables, 31</td>
</tr>
<tr>
<td>error codes, 187</td>
</tr>
<tr>
<td>installing applications and databases, 107</td>
</tr>
<tr>
<td>spatial function, ST_AsBinary, 373</td>
</tr>
<tr>
<td>spatial function, ST_AsExtText, 374</td>
</tr>
<tr>
<td>spatial function, ST_AsText, 374</td>
</tr>
<tr>
<td>spatial function, ST_Distance, 375</td>
</tr>
<tr>
<td>spatial function, ST_Equals, 375</td>
</tr>
<tr>
<td>spatial function, ST_IntersectsRect, 376</td>
</tr>
<tr>
<td>spatial function, ST_Point, 376</td>
</tr>
<tr>
<td>spatial function, ST_PointFromExtText, 377</td>
</tr>
<tr>
<td>spatial function, ST_PointFromText, 378</td>
</tr>
<tr>
<td>spatial function, ST_PointFromWKB, 378</td>
</tr>
<tr>
<td>spatial function, ST_SRID, 379</td>
</tr>
<tr>
<td>spatial function, ST_X, 379</td>
</tr>
<tr>
<td>spatial function, ST_Y, 380</td>
</tr>
<tr>
<td>SQL functions, aggregate, 294</td>
</tr>
<tr>
<td>SQL functions, data type conversion, 295</td>
</tr>
<tr>
<td>SQL functions, types of, 294</td>
</tr>
<tr>
<td>SQL statement reference, 395</td>
</tr>
<tr>
<td>SQLE_MAX_ROW_SIZE_EXCEEDED error, 46</td>
</tr>
<tr>
<td>supported network protocols, 3</td>
</tr>
<tr>
<td>system functions, 300</td>
</tr>
<tr>
<td>table owners, 251</td>
</tr>
<tr>
<td>timestamp_with_time_zone_format creation parameter, 153</td>
</tr>
<tr>
<td>troubleshooting, 461</td>
</tr>
<tr>
<td>users, 56</td>
</tr>
<tr>
<td>UTF8BIN collation for UNICODE characters, 25</td>
</tr>
<tr>
<td>utilities reference, 186</td>
</tr>
</tbody>
</table>

---

502 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
UltraLite file property, 177
UltraLite name property, 177
UltraLite schema upgrades reference, 397
UltraLite size limit, 8
unique keys, 54
upgrading schema on devices, 115
user IDs and passwords, 56
utilities reference, 186
validating, 41
viewing option settings, 35
Windows Mobile file paths, 34
UltraLite embedded SQL
(see also UltraLite embedded SQL library functions)
character strings, 192
line numbers, 192
preprocessor, 192
UltraLite engine
about, 16
concurrency in, 452
defining a temporary file location at connection
time, 175
defining executable location at connection time,
175
erase database utility, 196
error codes, 187
running as a daemon, 195
start utility, 195
stop utility, 196
troubleshooting, 461
troubleshooting error -764, 465
Windows Mobile deployment, 16
UltraLite engine start utility
syntax, 195
UltraLite engine stop utility (ulstop)
syntax, 196
UltraLite erase database utility
syntax, 196
UltraLite information utility (ulinfo)
syntax, 197
UltraLite initialize database utility (ulinit)
syntax, 198
UltraLite Java edition
about, 1
UltraLite Java Edition Database Information utility
(uljinfo)
syntax, 234
UltraLite Java Edition Database Load utility (uljload)
syntax, 235
UltraLite Java edition database options
about, 228
UltraLite Java edition database properties
about, 227
UltraLite Java Edition Database Transfer utility
(uljdbtserv)
delete database, 243
receive database using, 242
starting, 240
syntax, 239
transfer database using, 240
viewing database information, 243
viewing log file, 244
UltraLite Java Edition Database unload utility
(uljunload)
syntax, 237
UltraLite Java edition databases
(see also UltraLite Java edition management
system)
(see also UltraLiteJ)
columns, 245
describing columns of tables in, 248
describing foreign key columns in, 246
describing foreign keys in, 246
describing publications in, 244
describing tables in, 249
limitations, 8
storing indexes in, 247
storing properties of, 249
storing publications in, 248
uljdbtserv utility, 239
uljinfo utility, 234
uljload utility, 235
uljunload utility, 237
utilities, 234
UltraLite Java edition management system
(see also UltraLite Java edition databases)
(see also UltraLiteJ)
UltraLite load XML to database utility
syntax, 206
UltraLite optimizer
execution plan access options, 446
UltraLite passwords
about, 56
UltraLite plug-in
troubleshooting, 461
UltraLite publications
renaming during schema upgrade, 115
UltraLite runtime
about, 16
concurrency in, 452
UltraLite runtime libraries
Linux, 107
Windows, 107
Windows Mobile, 107
UltraLite schema
upgrading on device, 115
UltraLite SQL
(see also UltraLite embedded SQL)
comma-separated lists, 343
comments, 252
data types, 275
dates, 256
execution plans for, 446
expressions, 256
identifiers, 251
index-based optimizations, 441
keywords, 251
NULL values, 253
numbers, 253
operators, 272
page-based optimizations, 448
special values, 253
SQL functions date and time, 295
SQL functions, miscellaneous, 297
SQL functions, numeric, 298
SQL functions, string, 299
strings, 252
times, 256
troubleshooting queries, 464
tuning statement performance, 457
variables, 275
UltraLite SQL statements
about, 395
ALTER DATABASE SCHEMA FROM FILE
statement syntax, 397
ALTER PUBLICATION statement syntax, 398
ALTER SYNCHRONIZATION PROFILE
statement syntax, 399
ALTER TABLE statement syntax, 401
ALTER USER statement syntax, 404
categories of, 396
CHECKPOINT statement syntax, 405
COMMIT statement syntax, 406
CREATE INDEX statement syntax, 407
UNION operation
   UltraLite syntax, 436
 UNION statement
   UltraLite syntax, 436
union-all operation
   UltraLite execution plans, 450
unions
   UltraLite multiple select statements, 436
UNIQUE
   UltraLite CREATE INDEX parameter, 407
unique constraints
   UltraLite characteristics, 53
   UltraLite copying method, 51
   UltraLite CREATE TABLE statement, 412
   UltraLiteJ CREATE TABLE statement, 412
unique indexes
   UltraLite characteristics, 53
   UltraLite index creation from, 54
   UltraLite UNIQUE SQL parameter, 407
unique keys
   UltraLite characteristics, 53
   UltraLite index creation from, 54
UNIQUEIDENTIFIER data type
   UltraLite, 290
universally unique identifiers
   (see also UUIDs)
   UltraLite SQL syntax for NEWID function, 354
unloading
   UltraLite databases, 216
   UltraLite databases with ulunload, 216
unnamed foreign keys
   UltraLite usage, 412
   UltraLiteJ usage, 412
unpacked rows
   UltraLite about, 46
UPDATE statement
   UltraLite syntax, 437
updates
   UltraLite databases, 453
updating
   UltraLite updating rows, 437
UpgradeSchemaFromFile method
   UltraLite replacement for schema upgrade, 115
upgrading
   UltraLite schema error callback, 117
   UltraLite schema process, 115
   UltraLite SQL ALTER DATABASE SCHEMA
   FROM FILE syntax, 397
upgrading UltraLite
   troubleshooting connections, 461
upload ok
   UltraLite synchronization parameter, 101
upload only
   UltraLite synchronization parameter, 101
upload only synchronization
   UltraLite databases, 101
   UltraLite upload_only synchronization parameter, 101
upload_ok synchronization parameter
   UltraLite reference, 101
upload_only synchronization parameter
   UltraLite reference, 101
UPPER function
   UltraLite syntax, 390
uppercase characters
   UltraLite UPPER function, 390
uppercase strings
   UltraLite UCASE function, 389
   UltraLite UPPER function, 390
user authentication
   custom MobiLink, 85
   PWD UltraLite connection parameter, 172
   UltraLite, 56
   UltraLite Authentication Value synchronization parameter, 87
   UltraLite custom for synchronization, 91
   UltraLite getUsername method, 103
   UltraLite password synchronization parameter, 92
   UltraLite synchronization status reports, 86
   UltraLite user_name synchronization, 103
user data
   UltraLite synchronization parameter, 102
user IDs
   maximum length for UltraLite, 176
   UltraLite databases, 56
user name
   UltraLite synchronization parameter, 103
user-defined data types
   UltraLite equivalents, 293
   UltraLite unsupported in, 275
user_data
   UltraLite synchronization parameter, 102
user_name
   UltraLite synchronization parameter, 103
users
   creating for UltraLite databases, 61
managing in UltraLite databases, 56
UltraLite ALTER USER statement, 404
UltraLite DROP USER statement, 422
utf8_encoding creation parameter
UltraLite description, 155
utf8_encoding database property
UltraLite usage, 24
UTF8BIN collation
UltraLite considerations, 25
utilities
UltraLite, 186
UltraLite database unload (ulunload) syntax, 216
UltraLite engine start (uleng16) syntax, 195
UltraLite engine stop (ulstop) syntax, 196
UltraLite erase database (ulerase) syntax, 196
UltraLite error codes, 187
UltraLite information (ulinfo) syntax, 197
UltraLite initialize database (ulinit) syntax, 198
UltraLite Interactive SQL (dbisql) syntax, 187
UltraLite Java Edition Database Information (ulinfo) syntax, 234
UltraLite Java Edition Database Load (uljload) syntax, 235
UltraLite Java Edition Database Transfer utility (uljdtbserv) syntax, 239
UltraLite Java Edition Database Unload (uljunload) syntax, 237
UltraLite load XML to database (ulload) syntax, 206
UltraLite SQL preprocessor (sqlpp) syntax, 192
UltraLite synchronization (ulsync) syntax, 210
UltraLite troubleshooting, 464
UltraLite validate database (ulvalid) syntax, 219
Windows Mobile database administration on device, 34
UUIDs
UltraLite SQL syntax for NEWID function, 354
UltraLite SQL syntax for STRTOUUID function, 382
UltraLite SQL syntax for UUIDTOSTR function, 391
UUIDTOSTR function
UltraLite syntax, 391

V
validate database utility (ulvalid)
   UltraLite about, 219
validating
   (see also validating databases)
   UltraLite checksum_level creation parameter, 134
   UltraLite databases, 41
   UltraLite databases with ulvalid, 219
validating databases
   UltraLite validate database utility (ulvalid), 219
values
   UltraLite index hash, 141
VARBINARY data type
   UltraLite, 291
VARCHAR data type
   UltraLite, 276
variables
   UltraLite SQL, 275
version
   UltraLite synchronization parameter, 104
version synchronization parameter
   UltraLite reference, 104
versions
   UltraLite troubleshooting utilities, 464
viewing
   UltraLite execution plans, 449
   UltraLite table methods, 50
viewing UltraLite database settings
about, 35
virtual file system (see VFS)
Visual Studio
   UltraLite building CustDB application, 15

W
WEEKS function
   UltraLite syntax, 392
WHEN
   UltraLite CASE expression, 259
WHERE clause
   CREATE PUBLICATION statement [MobiLink][SQL Remote], 409
   UltraLite CREATE PUBLICATION statement, 409
   UltraLite DELETE statement, 418
   UltraLite publication usage, 79
   UltraLite search conditions, 263
   UltraLite SELECT statement, 430
   UltraLite synchronization limitations, 72
   UltraLite UPDATE statement, 438
   whole tables
UltraLite publishing, 72
wildcards
  UltraLite LIKE search condition, 270
  UltraLite PATINDEX function, 356

Windows
  (see also Windows ME)
  (see also Windows Mobile 5)
  (see also Windows Vista)
  (see also Windows XP/200x)
  UltraLite character sets, 25
  UltraLite runtime libraries for, 107

Windows desktop
  UltraLite databases, 34

Windows Mobile
  choosing an API, 16
  troubleshooting error -764, 465
  UltraLite ActiveSync deployment, 118
  UltraLite building CustDB application using .NET, 15
  UltraLite character sets, 25
  UltraLite considerations, 15
  UltraLite engine and runtime support, 16
  UltraLite engine deployment, 16
  UltraLite file path prefix, 34
  UltraLite FIPS enablement, 140
  UltraLite MobiLink clients, 81
  UltraLite runtime libraries for, 107
  UltraLite uleng16 deployment, 16

WITH CHECKPOINT clause
  UltraLite LOAD TABLE statement, 427

words
  UltraLite keywords, 251
  UltraLite reserved words, 251

working with indexes
  UltraLite about, 52

X

XML
  loading to database, 206
  sourcing UltraLite databases from, 21
  UltraLite unloading databases, 210

XML data type
  UltraLite equivalent, 293

Y

YEAR function
  UltraLite syntax, 393

YEARS function
  UltraLite syntax, 394

YMD function
  UltraLite syntax, 395

Z

zero-padding
  controlling with
  timestamp_with_time_zone_format option, 155
  UltraLite date_format creation parameter, 137
  UltraLite timestamp_format creation parameter, 152