



UltraLite® Database Management and Reference

Published: October 2006

Copyright and trademarks

Copyright © 2006 iAnywhere Solutions, Inc. Portions copyright © 2006 Sybase, Inc. All rights reserved.

iAnywhere Solutions, Inc. is a subsidiary of Sybase, Inc.

iAnywhere grants you permission to use this document for your own informational, educational, and other non-commercial purposes; provided that (1) you include this and all other copyright and proprietary notices in the document in all copies; (2) you do not attempt to "pass-off" the document as your own; and (3) you do not modify the document. You may not publish or distribute the document or any portion thereof without the express prior written consent of iAnywhere.

This document is not a commitment on the part of iAnywhere to do or refrain from any activity, and iAnywhere may change the content of this document at its sole discretion without notice. Except as otherwise provided in a written agreement between you and iAnywhere, this document is provided "as is", and iAnywhere assumes no liability for its use or any inaccuracies it may contain.

iAnywhere®, Sybase®, and the marks listed at <http://www.iAnywhere.com/trademarks> are trademarks of Sybase, Inc. or its subsidiaries. ® indicates registration in the United States of America.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Contents

About This Manual	vii
SQL Anywhere documentation	viii
Documentation conventions	xi
Finding out more and providing feedback	xv
I. Introducing UltraLite	1
Introducing UltraLite	3
Introducing the UltraLite database management system	4
Comparing UltraLite and SQL Anywhere	5
UltraLite-specific decisions you need to make	10
Understanding database management fundamentals for UltraLite	15
II. Using UltraLite Databases	21
Creating and Configuring UltraLite Databases	23
Creating UltraLite databases	24
Choosing creation-time database properties	30
Configuring post-creation database options	40
Connecting to an UltraLite Database	43
Introducing UltraLite database connections	44
Opening connections with connection strings	47
Storing parameters with the ULSQLCONNECT environment variable	53
Working with UltraLite Databases	55
Working with UltraLite tables and columns	56
Working with UltraLite indexes	65
Working with UltraLite publications	72
Working with UltraLite users	76
Viewing UltraLite database settings	78
Exploring the CustDB Samples for UltraLite	79
Introducing CustDB	80
Finding CustDB sample files	81
Lesson 1: Logging in and populating the UltraLite remote	83

Lesson 2: Using the CustDB client application 84
 Lesson 3: Synchronizing with the CustDB consolidated database 86
 Lesson 4: Browsing MobiLink synchronization scripts 88
 What's next? 90

III. UltraLite Database Reference 91

UltraLite Database Settings Reference 93

case property 94
 checksum_level property 95
 date_format property 97
 date_order property 100
 fips property 102
 global_id option 104
 max_hash_size property 106
 ml_remote_id option 108
 nearest_century property 109
 obfuscate property 110
 page_size property 112
 precision property 114
 scale property 116
 time_format property 118
 timestamp_format property 120
 timestamp_increment property 123
 utf8_encoding property 125

UltraLite Connection String Parameters Reference 127

CACHE_SIZE connection parameter 128
 CON connection parameter 130
 DBF connection parameter 131
 CE_FILE connection parameter 133
 NT_FILE connection parameter 135
 PALM_FILE connection parameter 137
 PALM_DB connection parameter 139
 SYMBIAN_FILE connection parameter 140
 DBN connection parameter 142
 DBKEY connection parameter 143

PALM_ALLOW_BACKUP connection parameter	145
PWD connection parameter	146
RESERVE_SIZE connection parameter	148
START connection parameter	149
UID connection parameter	150
UltraLite Utilities Reference	153
Introduction to UltraLite utilities	154
Interactive SQL utility (dbisql)	155
SQL Preprocessor for UltraLite utility (sqlpp)	158
UltraLite AppForge Registry utility (ulafreg)	161
UltraLite HotSync Conduit Installation utility for Palm OS (ulcond10)	163
UltraLite Create Database utility (ulcreate)	165
UltraLite Engine utility (uleng10)	168
UltraLite Information utility (ulinfo)	169
UltraLite Initialize Database utility (ulinit)	172
UltraLite Load XML to Database utility (ulload)	174
UltraLite Engine Stop utility (ulstop)	177
UltraLite Synchronization utility (ulsync)	178
UltraLite Unload Database to XML utility (ulunload)	180
UltraLite Unload Old Database utility (ulunloadold)	183
UltraLite Data Management utility for Palm OS (ULDBUtil)	185
Supported extended options	186
Supported exit codes	189
UltraLite System Table Reference	191
UltraLite system tables	192
IV. UltraLite SQL Reference	199
UltraLite SQL Elements Reference	201
Keywords in UltraLite	202
Identifiers in UltraLite	203
Strings in UltraLite	204
Comments in UltraLite	205
Numbers in UltraLite	206
The NULL value in UltraLite	207
Special values in UltraLite	208

Dates and times in UltraLite	211
Data types in UltraLite	212
Expressions in UltraLite	215
Operators in UltraLite	228
Variables in UltraLite	231
Query access plans in UltraLite	232
UltraLite SQL Function Reference	235
Function types	236
Alphabetical list of functions	241
UltraLite SQL Statement Reference	319
UltraLite SQL statements overview	320
ALTER TABLE statement	322
ALTER PUBLICATION statement	326
COMMIT statement	327
CREATE INDEX statement	328
CREATE PUBLICATION statement	330
CREATE TABLE statement	331
DELETE statement	336
DROP INDEX statement	337
DROP PUBLICATION statement	338
DROP TABLE statement	339
INSERT statement	340
ROLLBACK statement	341
SELECT statement	342
START SYNCHRONIZATION DELETE statement	347
STOP SYNCHRONIZATION DELETE statement	348
TRUNCATE TABLE statement	349
UNION operation	351
UPDATE statement	352
Index	353

About This Manual

Subject

This manual introduces the UltraLite database system for small devices.

Audience

This manual is intended for all developers who want to take advantage of the performance, resource efficiency, robustness, and security of an UltraLite relational database for data storage and synchronization for embedded or mobile devices.

SQL Anywhere documentation

This book is part of the SQL Anywhere documentation set. This section describes the books in the documentation set and how you can use them.

The SQL Anywhere documentation

The complete SQL Anywhere documentation is available in two forms: an online form that combines all books, and as separate PDF files for each book. Both forms of the documentation contain identical information and consist of the following books:

- ◆ **SQL Anywhere 10 - Introduction** This book introduces SQL Anywhere 10—a comprehensive package that provides data management and data exchange, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.
- ◆ **SQL Anywhere 10 - Changes and Upgrading** This book describes new features in SQL Anywhere 10 and in previous versions of the software.
- ◆ **SQL Anywhere Server - Database Administration** This book covers material related to running, managing, and configuring SQL Anywhere databases. It describes database connections, the database server, database files, security, backup procedures, security, and replication with Replication Server, as well as administration utilities and options.
- ◆ **SQL Anywhere Server - SQL Usage** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- ◆ **SQL Anywhere Server - SQL Reference** This book provides a complete reference for the SQL language used by SQL Anywhere. It also describes the SQL Anywhere system views and procedures.
- ◆ **SQL Anywhere Server - Programming** This book describes how to build and deploy database applications using the C, C++, and Java programming languages, as well as Visual Studio .NET. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools.
- ◆ **SQL Anywhere 10 - Error Messages** This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.
- ◆ **MobiLink - Getting Started** This manual introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.
- ◆ **MobiLink - Server Administration** This manual describes how to set up and administer MobiLink applications.
- ◆ **MobiLink - Client Administration** This manual describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases.
- ◆ **MobiLink - Server-Initiated Synchronization** This manual describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization or other remote actions from the consolidated database.

- ◆ **QAnywhere** This manual describes QAnywhere, which defines a messaging platform for mobile and wireless clients as well as traditional desktop and laptop clients.
- ◆ **SQL Remote** This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.
- ◆ **SQL Anywhere 10 - Context-Sensitive Help** This manual provides context-sensitive help for the Connect dialog, the Query Editor, the MobiLink Monitor, the SQL Anywhere Console utility, the Index Consultant, and Interactive SQL.
- ◆ **UltraLite - Database Management and Reference** This manual introduces the UltraLite database system for small devices.
- ◆ **UltraLite - AppForge Programming** This manual describes UltraLite for AppForge. With UltraLite for AppForge you can develop and deploy database applications to handheld, mobile, or embedded devices, running Palm OS, Symbian OS, or Windows CE.
- ◆ **UltraLite - .NET Programming** This manual describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.
- ◆ **UltraLite - M-Business Anywhere Programming** This manual describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows CE, or Windows XP.
- ◆ **UltraLite - C and C++ Programming** This manual describes UltraLite C and C++ programming interfaces. With UltraLite you can develop and deploy database applications to handheld, mobile, or embedded devices.

Documentation formats

SQL Anywhere provides documentation in the following formats:

- ◆ **Online documentation** The online documentation contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools. The online documentation is updated with each maintenance release of the product, and is the most complete and up-to-date source of documentation.

To access the online documentation on Windows operating systems, choose Start ► Programs ► SQL Anywhere 10 ► Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

To access the online documentation on Unix operating systems, see the HTML documentation under your SQL Anywhere installation or on your installation CD.

- ◆ **PDF files** The complete set of SQL Anywhere books is provided as a set of Adobe Portable Document Format (pdf) files, viewable with Adobe Reader.

On Windows, the PDF books are accessible from the online books via the PDF link at the top of each page, or from the Windows Start menu (Start ► Programs ► SQL Anywhere 10 ► Online Books - PDF Format).

On Unix, the PDF books are accessible on your installation CD.

Documentation conventions

This section lists the typographic and graphical conventions used in this documentation.

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- ◆ **Keywords** All SQL keywords appear in uppercase, like the words ALTER TABLE in the following example:

```
ALTER TABLE [ owner.]table-name
```

- ◆ **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown like the words *owner* and *table-name* in the following example:

```
ALTER TABLE [ owner.]table-name
```

- ◆ **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots), like *column-constraint* in the following example:

```
ADD column-definition [ column-constraint, ... ]
```

One or more list elements are allowed. In this example, if more than one is specified, they must be separated by commas.

- ◆ **Optional portions** Optional portions of a statement are enclosed by square brackets.

```
RELEASE SAVEPOINT [ savepoint-name ]
```

These square brackets indicate that the *savepoint-name* is optional. The square brackets should not be typed.

- ◆ **Options** When none or only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in square brackets.

```
[ ASC | DESC ]
```

For example, you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

- ◆ **Alternatives** When precisely one of the options must be chosen, the alternatives are enclosed in curly braces and a bar is used to separate the options.

```
[ QUOTES { ON | OFF } ]
```

If the QUOTES option is used, one of ON or OFF must be provided. The brackets and braces should not be typed.

File name conventions

The documentation generally adopts Windows conventions when describing operating-system dependent tasks and features such as paths and file names. In most cases, there is a simple transformation to the syntax used on other operating systems.

- ◆ **Directories and path names** The documentation typically lists directory paths using Windows conventions, including colons for drives and backslashes as a directory separator. For example,

```
MobiLink\redirector
```

On Unix, Linux, and Mac OS X, you should use forward slashes instead. For example,

```
MobiLink/redirector
```

- ◆ **Executable files** The documentation shows executable file names using Windows conventions, with the suffix *.exe*. On Unix, Linux, and Mac OS X, executable file names have no suffix. On NetWare, executable file names use the suffix *.nlm*.

For example, on Windows, the network database server is *dbsrv10.exe*. On Unix, Linux, and Mac OS X, it is *dbsrv10*. On NetWare, it is *dbsrv10.nlm*.

- ◆ **install-dir** The installation process allows you to choose where to install SQL Anywhere, and the documentation refers to this location using the convention *install-dir*.

After installation is complete, the environment variable `SQLANY10` specifies the location of the installation directory containing the SQL Anywhere components (*install-dir*). `SQLANYSH10` specifies the location of the directory containing components shared by SQL Anywhere with other Sybase applications.

For more information on the default location of *install-dir*, by operating system, see “[File Locations and Installation Settings](#)” [*SQL Anywhere Server - Database Administration*].

- ◆ **samples-dir** The installation process allows you to choose where to install the samples that are included with SQL Anywhere, and the documentation refers to this location using the convention *samples-dir*.

After installation is complete, the environment variable `SQLANYXSAMP10` specifies the location of the directory containing the samples (*samples-dir*). From the Windows Start menu, choosing Programs ► SQL Anywhere 10 ► Sample Applications and Projects opens a Windows Explorer window in this directory.

For more information on the default location of *samples-dir*, by operating system, see “[The samples directory](#)” [*SQL Anywhere Server - Database Administration*].

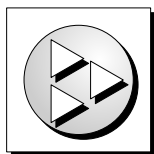
- ◆ **Environment variables** The documentation refers to setting environment variables. On Windows, environment variables are referred to using the syntax *%envvar%*. On Unix, Linux, and Mac OS X, environment variables are referred to using the syntax *\$envvar* or *\${envvar}*.

Unix, Linux, and Mac OS X environment variables are stored in shell and login startup files, such as *.cshrc* or *.tcshrc*.

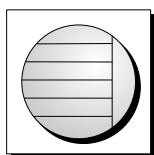
Graphic icons

The following icons are used in this documentation.

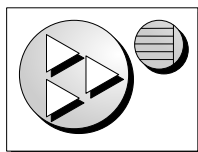
- ◆ A client application.



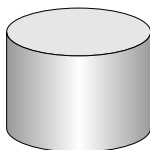
- ◆ A database server, such as SQL Anywhere.



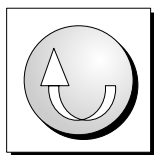
- ◆ An UltraLite application.



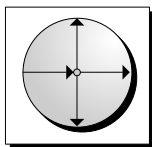
- ◆ A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



- ◆ Replication or synchronization middleware. These assist in sharing data among databases. Examples are the MobiLink server and the SQL Remote Message Agent.



- ◆ A Sybase Replication Server



- ◆ A programming interface.



Finding out more and providing feedback

Finding out more

Additional information and resources, including a code exchange, are available at the iAnywhere Developer Network at <http://www.ianywhere.com/developer/>.

If you have questions or need help, you can post messages to the iAnywhere Solutions newsgroups listed below.

When you write to one of these newsgroups, always provide detailed information about your problem, including the build number of your version of SQL Anywhere. You can find this information by entering **dbeng10 -v** at a command prompt.

The newsgroups are located on the *forums.sybase.com* news server. The newsgroups include the following:

- ◆ [sybase.public.sqlanywhere.general](#)
- ◆ [sybase.public.sqlanywhere.linux](#)
- ◆ [sybase.public.sqlanywhere.mobilink](#)
- ◆ [sybase.public.sqlanywhere.product_futures_discussion](#)
- ◆ [sybase.public.sqlanywhere.replication](#)
- ◆ [sybase.public.sqlanywhere.ultralite](#)
- ◆ [ianywhere.public.sqlanywhere.qanywhere](#)

Newsgroup disclaimer

iAnywhere Solutions has no obligation to provide solutions, information, or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and ensure its operation and availability.

iAnywhere Solutions Technical Advisors as well as other staff assist on the newsgroup service when they have time available. They offer their help on a volunteer basis and may not be available on a regular basis to provide solutions and information. Their ability to help is based on their workload.

Feedback

We would like to receive your opinions, suggestions, and feedback on this documentation.

You can email comments and suggestions to the SQL Anywhere documentation team at iasdoc@ianywhere.com. Although we do not reply to emails sent to that address, we read all suggestions with interest.

In addition, you can provide feedback on the documentation and the software through the newsgroups listed above.

Part I. Introducing UltraLite

This part introduces the UltraLite relational database system for small devices. It describes general features of the UltraLite database, and summarizes some of the internal mechanisms UltraLite uses to manage data and transactions.

CHAPTER 1

Introducing UltraLite

Contents

Introducing the UltraLite database management system 4

Comparing UltraLite and SQL Anywhere 5

UltraLite-specific decisions you need to make 10

Understanding database management fundamentals for UltraLite 15

About this chapter

This chapter introduces you to the UltraLite database management system. Use this chapter to choose whether or not you can use UltraLite address your business needs, and how to choose the appropriate development API to help you achieve your goals—particularly in a multi-platform deployment scenario.

Introducing the UltraLite database management system

UltraLite is a relational database similar to SQL Anywhere—but much smaller in scale. UltraLite is a mobile database with true platform independence, and is designed to create custom solutions for small-footprint devices such as cell phones, personal organizers, and embedded devices.

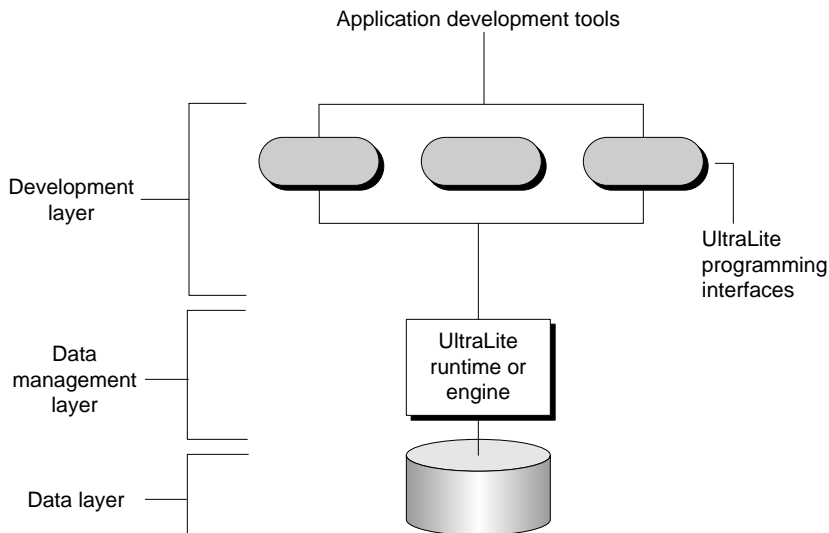
The UltraLite database is the foundation of the UltraLite solution. Beyond just a database, however, UltraLite provides you with a complete database management system:

- ◆ **A development layer** UltraLite supports various programming interfaces that keep you from getting locked into one deployment platform, development tool, or set of IT infrastructure products. For information about which API you should choose, see [“Choosing your programming interface” on page 12](#).

To help you maintain your UltraLite project, UltraLite completes its development support with a comprehensive set of administration tools that you can run either from a command line or from graphical administration tools like UltraLite wizards in Sybase Central or Interactive SQL.

- ◆ **A data management layer** You can connect to an UltraLite database, either with an in-process library (also called a runtime), or a separate process called an engine. Both processes control connection and data access requests. They also include a built-in bi-directional synchronization framework that links field and mobile workers with enterprise back-end systems. For information about which process you should choose, see [“Choosing a data management component” on page 11](#).

The data, management, and development layers are represented in the following figure.



Comparing UltraLite and SQL Anywhere

Certain features or metrics may help you determine which database type is best suited to your deployments.

The footprint of an UltraLite database and application are very small—typically in the 400 KB range for the C/C++ version, compared to the approximately 6 MB footprint required by a SQL Anywhere database server and synchronization client. Therefore, while UltraLite does not support some of the features included with the SQL Anywhere (for example, triggers and stored procedures), many of the basic features of a relational database are included.

Contrasting SQL Anywhere features with UltraLite

If you are unsure which database you require, compare supported features of SQL Anywhere and UltraLite. By comparing feature sets, you can better evaluate the suitability of each database.

Feature required	SQL Anywhere	UltraLite	Considerations
Transaction logs	X		UltraLite does not use an external transaction log. Instead, it employs a unique transaction processing mechanism. See “Transaction processing, recovery, and backup” on page 19 . Note: If you decide to deploy SQL Anywhere on a small footprint device, remember to manage transaction logs carefully, as transaction logs grow quickly.
Transaction processing, referential integrity, and multi-table joins	X	X	
Triggers, stored procedures, and views	X		
External stored procedures (callable external DLLs)	X		
Built-in referential and entity integrity, including cascading updates and deletes	X	Limited	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. See “Avoiding synchronization issues with foreign key cycles” [MobiLink - Client Administration] and “Table Order synchronization parameter” [MobiLink - Client Administration] .
Dynamic, multiple database support	X	X	
Multi-threaded application support	X	X	

Feature required	SQL Anywhere	UltraLite	Considerations
Row-level locking	X	X	
XML unload and load utilities		X	UltraLite uses separate administration tools to accomplish XML load and unloads. It is not built into the runtime. See “UltraLite Load XML to Database utility (uload)” on page 174 and “UltraLite Unload Database to XML utility (ulunload)” on page 180.
XML export and import utilities	X		SQL Anywhere uses SQL statements to export/import data to XML. You can also use dbunload to export your data. See “Importing and Exporting Data” [<i>SQL Anywhere Server - SQL Usage</i>].
SQLX functionality	X		
SQL functions	X	Limited	Not all SQL functions are available for use in UltraLite applications. Using an unsupported function gives a <code>Feature not available in UltraLite</code> error. For a complete list of supported functions, see “UltraLite SQL Function Reference SQL Functions” on page 235.
SQL statements	X	Limited	The scope of SQL statements are limited in UltraLite. For a complete list of supported statements, see “UltraLite SQL Statement Reference” on page 319.
Integrated HTTP server	X		
Strong encryption for database files and network communications	X	X	
Event scheduling and handling	X		
High-performance, self-tuning, cost-based query optimizer	X		UltraLite has a query optimizer, but it is not as extensive as that of SQL Anywhere. Therefore, the UltraLite optimizer may not provide as high performance as the SQL Anywhere optimizer on complex queries. However if you are running simple queries, UltraLite can run faster. See “Query access plans in UltraLite” on page 232.
Choice of several thread-safe APIs	X	X	UltraLite gives application developers a uniquely flexible architecture that will allow for the creation of applications for changing and/or varied deployment environments. See “Choosing your programming interface” on page 12.

Feature required	SQL Anywhere	UltraLite	Considerations
Cursor support	X	X	
Dynamic cache sizing with an advanced cache management system	X		Cache sizing is static in UltraLite. Nonetheless, UltraLite allows you to set the cache size when the database is started, which gives you the ability to scale cache size accordingly. See “CACHE_SIZE connection parameter” on page 128 .
Database recovery after system or application failure	X	X	
Binary Large Object (BLOB) support	X	X	UltraLite cannot index or compare BLOBs.
Windows Performance Monitor integration	X		
Online table and index defragmentation	X		
Online backup	X		
Small footprint, which can be as small as 500 KB		X	Small footprint devices tend to have relatively slow processors. UltraLite employs algorithms and data structures that are targeted for these devices, so UltraLite continues to provide high performance and low memory use.
Runs on smart phones		X	
Direct device connections to a Windows CE device from the desktop.		X	SQL Anywhere databases need a server before allowing desktop connections to the database that you deploy on a Windows CE device. On UltraLite, you simply need to prefix the connection string with WCE:\ . See “Windows CE” on page 51 .
High-performance updates and retrievals through use of indexes	X	X	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 “Index performance considerations” on page 33 .
Synchronizing to Oracle, DB2, Sybase Adaptive Server Enterprise, or SQL Anywhere	X	X	

Feature required	SQL Anywhere	UltraLite	Considerations
Built-in synchronization		X	Unlike SQL Anywhere deployments, UltraLite does not require a client agent to facilitate synchronization. Synchronization is built into the UltraLite runtime to minimize the components you need to deploy. See “UltraLite Clients” [MobiLink - Client Administration] .
In-process runtime support in UltraLite APIs		X	
Computed columns	X		
Declared temporary tables/ Global temporary tables	X		
System functions	X		UltraLite does not support SQL Anywhere system functions, including property functions. You cannot include them as part of your UltraLite application.
Timestamp columns	X	X	SQL Anywhere Transact-SQL timestamp columns are created with the DEFAULT TIMESTAMP default. UltraLite timestamp columns are created with the DEFAULT CURRENT TIMESTAMP default. Therefore, UltraLite does not automatically update the timestamp when the row is updated.
User-based permission scheme to determine object-based ownership and access	X		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 “The role of user authentication” on page 45 .

UltraLite size and number limitations

In most cases, the memory, CPU, and storage required by most mobile devices and embedded systems impose stricter limits that typically make UltraLite the preferred database option. The following table lists the absolute database size and object limitations imposed by UltraLite data structures.

 To compare UltraLite limitations with SQL Anywhere limitations, see [“Size and number limitations” \[SQL Anywhere Server - Database Administration\]](#).

Statistic	Maximum for UltraLite
Number of connections per database	Up to 14.

Statistic	Maximum for UltraLite
File-based persistent store (database size)	2 GB file or OS limit on file size.
Palm Computing Platform database size	128 MB (primary storage). 2 GB (expansion card file system).
Rows per table	Up to 16 million.
Row ID size	3 bytes.
Rows per database	Limited by persistent store.
Table size	Limited only by database size.
Tables per database	Limited only by database size.
Columns per table	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 will be much less than 4000.
Indexes per table	Limited only by database size.
Tables referenced per transaction	No limit.
Stored procedure length	Not applicable.
Stored procedures per database	Not applicable.
Triggers per database	Not applicable.
Nesting	Not applicable.
Number of publications	32 publications.
Page size	Up to 16 KB.
Row size	The length of each stored row must be less than the page size. Character strings are stored without padding when they are shorter than the column size. This excludes columns declared as long binary and long varchar as these are stored separately.
Long binary/long varchar size	Limited only by database size.
Prepared statement size	The size of a prepared statement, which cannot exceed 64 KB. The size includes the declared size of each new row referenced by the sql query result set as well as table rows at their declared size.

UltraLite-specific decisions you need to make

If you decide to implement an UltraLite solution, you need to think about:

- ◆ How flexible or scalable do you require your deployment to be? See [“Planning for scalability” on page 10](#).
- ◆ How many applications need to connect to the UltraLite database? The number of concurrent connections affects whether you need the UltraLite in-process runtime or the UltraLite engine. See [“Choosing a data management component” on page 11](#) to understand how they differ.
- ◆ On which platforms will the database run on? Because file formats have been consolidated, you may be able to create a database that runs on multiple platforms. See [“Choosing an UltraLite deployment environment” on page 12](#) and [“Understanding database management fundamentals for UltraLite” on page 15](#).

Tip

If you need to create a file format that suits multiple platforms, use the Create Database wizard in Sybase Central to help you determine whether or not this is possible.

- ◆ What platform(s) do you want to support? This can affect which APIs are available to program your application. See [“Choosing your programming interface” on page 12](#).

Planning for scalability

If you require the compact size of an UltraLite database, but don't want to sacrifice full database functionality, implement an UltraLite solution on devices with limited resources and then synchronize to a consolidated database using MobiLink synchronization technology.

UltraLite developers can synchronize the data in UltraLite databases with a central consolidated database like SQL Anywhere. This consolidated database may be a desktop database for personal applications, or a multi-user database for shared enterprise data. This mixture allows you to scale your solution as need arises.

Supported network protocols

Review the following table to determine which platforms support any given synchronization streams/network protocols.

Network protocol	Windows desktop (32-bit)	Embedded Windows XP	Windows CE	Palm OS	Symbian OS
TCP/IP synchronization	X	X	X	X	X
HTTP synchronization	X	X	X	X	X
HTTPS synchronization	X	X	X	X	

Network protocol	Windows desktop (32-bit)	Embedded Windows XP	Windows CE	Palm OS	Symbian OS
Transport-layer security (TLS) over HTTP or TCP/IP synchronization	X	X	X ¹	X	
HotSync synchronization				X	
ActiveSync synchronization (3.5 and later)			X ²		

¹ Pocket PC required.

² For UltraLite.NET only. Not available for Smartphone 2002.

See also

- ◆ “UltraLite Clients” [[MobiLink - Client Administration](#)]
- ◆ “Network protocol options for UltraLite synchronization streams” [[MobiLink - Client Administration](#)]
- ◆ “Introducing MobiLink Synchronization” [[MobiLink - Getting Started](#)]
- ◆ “UltraLite Synchronization utility (ulsync)” on page 178

Choosing a data management component

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 library types:

- ◆ **UltraLite in-process runtime library (*ulrt*.**)** In UltraLite, the runtime and the application are part of the same process, which makes the database specific to the application. For all platforms, the runtime manages UltraLite databases as well as built-in synchronization operations. The UltraLite runtime can manage a maximum of four databases at any one time, and is based on a single code base irrespective of the platform used.
- ◆ **UltraLite database engine client library (*ulrtc*.**)** For Windows desktop and Windows CE platforms, a separate executable exists that provides access to the database from multiple applications. This executable is supported for the following APIs: UltraLite C/C++, UltraLite for AppForge, UltraLite for M-Business Anywhere, and UltraLite.NET.

Each application must use a client library when using the UltraLite engine. This client library allows each application to communicate with the UltraLite engine. The UltraLite engine requires more system resources than the UltraLite runtime and may yield lower performance.

☞ For a comparison of UltraLite with SQL Anywhere databases, see “[Comparing UltraLite and SQL Anywhere](#)” on page 5.

See also

- ◆ UltraLite for AppForge: [UltraLite - AppForge Programming](#) [[UltraLite - AppForge Programming](#)]

- ◆ UltraLite.NET: [UltraLite - .NET Programming \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite C/C++ and embedded SQL: [UltraLite - C and C++ Programming \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [UltraLite - M-Business Anywhere Programming \[UltraLite - M-Business Anywhere Programming\]](#)

Choosing an UltraLite deployment environment

You can deploy UltraLite to various platforms, however, certain tools are only available to certain platforms.

 For a list of supported libraries, see “[Choosing your programming interface](#)” on page 12.

Component	Windows desktop (32-bit)	Embedded Windows XP	Windows CE	Palm OS	Symbian OS
UltraLite database administration tools (utilities and wizards)	X	X			
Embedded SQL preprocessor (sqlpp)	X	X			
UltraLite engine	X		X	¹	
Custom UltraLite client applications	X	X	X	X	X

¹ Because Palm OS exclusively runs one application at a time, the engine is not required on this platform.

Choosing your programming interface

All UltraLite APIs expose core database functionality through a variety of APIs. Some of the following APIs integrate with the development environment to simplify programming tasks.

- ◆ C/C++ interface
- ◆ Embedded SQL for C/C++
- ◆ UltraLite.NET
- ◆ AppForge Crossfire for C# or VB.NET
- ◆ M-Business Anywhere for JavaScript

UltraLite APIs offer different data access libraries 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 that will allow for the creation of applications for varied deployment environments.

Note

Pocket Builder is not supported in this version of UltraLite. Sybase PocketBuilder is not included with SQL Anywhere. Contact Sybase for details (<http://www.sybase.com/products/developmentintegration/pocketbuilder>).

See also

- ◆ UltraLite for AppForge: [UltraLite - AppForge Programming \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite.NET: [UltraLite - .NET Programming \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite C/C++ and embedded SQL: [UltraLite - C and C++ Programming \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [UltraLite - M-Business Anywhere Programming \[UltraLite - M-Business Anywhere Programming\]](#)

◆ To choose your programming interface

1. Choose your target platform(s). UltraLite supports Palm OS, Windows CE, Windows XP/embedded Windows XP, and Symbian OS.
2. For each platform you need to support, determine if the API supports that platform. Different APIs support different platforms. If you are doing cross-platform development, choose an API that supports more than one intended target.

The correlation of step 1 and 2 is captured in the a table that follows. Use this support matrix to help you quickly identify your development options.

Deployment targets	UltraLite App-Forge Crossfire for MobileVB or C# ¹	UltraLite for C/ C++ and embedded SQL	UltraLite.NET ²	UltraLite for M-Business Anywhere ³
Palm OS	version 4 +	version 4 +	N/A	version 5.0 +
Windows CE	CE 3.0 +	On CE 3.0 +	CE 3.0 +, with .NET compact framework 1.0.3705 CE 5.0 with .NET compact framework 2.0	version 3.0 + for Pocket PC
Embedded Windows XP	N/A	Supported	.NET compact framework 1.0.5000	N/A
Symbian OS	version 7.0/8.0	version 7.0/8.0	N/A	N/A

¹ Development for AppForge MobileVB as an extension to Microsoft Visual Basic or to Microsoft Visual Studio.NET.

² Development as an extension to Microsoft Visual Studio.NET. The driver supports ADO.NET versions 1.0 and 2.0.

³ For browser-based deployments of UltraLite programming in JavaScript.

3. Consider the effects of the following, and then finalize your selection accordingly:

SQL Anywhere compatibility If database compatibility with SQL Anywhere is a concern, consider the following:

- ◆ SQL Anywhere embedded SQL support provides a common programming interface for UltraLite and SQL Anywhere databases.
- ◆ ADO.NET provides common programming models that are shared between UltraLite components and SQL Anywhere.

Maintaining a common interface may be particularly useful on platforms such as Windows CE, where both databases are available. If you need to move from UltraLite to the more powerful and full-featured SQL Anywhere database, using embedded SQL or ADO.NET makes migrating your application easier.

Tip

Even though a common interface exists, it is still good practice to create an abstract data-access layer when writing your application.

Simplified deployments If simplifying your UltraLite deployment is an issue, consider programming with the M-Business Anywhere API. Your end-users can download both the UltraLite application and the database concurrently.

Application size and performance If creating a small application footprint is a concern, you should program your application with the C/C++ API. These applications typically yield the best performance and still maintain a small application file size because they do not interact with any software—other than the device platform.

Understanding database management fundamentals for UltraLite

This section describes file and data management features of both in-process runtime and the engine, so that you can understand what impact these features have on application development.

The UltraLite database file

A device's file management requirements dictate how an UltraLite database is stored and what database name conventions must be adhered to. While most platforms use traditional file-based storage, others, like the Palm OS record-based store, require that the database be saved differently. However, for the sake of simplicity, the UltraLite database is typically referred to as a file. Depending on platform restrictions, you may be able to create your device on the desktop, and then deploy the same file to one or more platforms.

☞ For more information about file name conventions used by different platforms when creating UltraLite database names, see [“Specifying file paths in a connection parameter” on page 50](#).

The UltraLite database schema

The logical framework of the database is known as a **schema**. In UltraLite, the schema is maintained as a catalog of system tables that hold the metadata for the UltraLite database. Metadata stored in the system tables include:

- ◆ Index definitions. See [“sysindex system table” on page 194](#) and [“sysixcol system table” on page 195](#).
- ◆ Table definitions. See [“systable system table” on page 192](#).
- ◆ Column definitions. See [“syscolumn system table” on page 193](#).
- ◆ Publication definitions. See [“syspublication system table” on page 196](#) and [“sysarticle system table” on page 196](#).
- ◆ User names and passwords. See [“sysuldata system table” on page 197](#).

Impact of schema changes in UltraLite

You can change the schema of a database with Data Definition Language (DDL) statements. A schema change can take a considerable amount of time; for example, changing the type of a column means that all rows in the associated table must be updated. Therefore, DDL statements only successfully execute when there aren't any:

- ◆ Uncommitted transactions.
- ◆ Other active uses of the schema (for example, synchronization, prepared but unreleased statements, executing database operations).

If either of these conditions is true, the DDL statement fails. When the DDL statement is executing, any other attempt to use the database will be blocked until the DDL statements completes the schema change.

☞ For a list of DDL statements supported by UltraLite, see [“Statement categories” on page 320](#).

The temporary file

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.

Temporary files are only used by UltraLite and are maintained in the same file path (if one exists) as the UltraLite database itself. The temporary file has the same file name as the database you have created, but with the following difference:

- ◆ **For file-based platforms** The tilde is included in the extension of the file (that is, *~db*). For example, if you run the *CustDB.udb* sample database, you'll notice the temporary file called *CustDB.~db* is maintained in the same folder as this file.
- ◆ **For record-based platforms** The tilde is appended to the end of the name of the file. For example, if *CustDB.udb* existed as a record-based file for Palm OS, the temporary file for it would be maintained as *CustDB.udb~*.

This temporary file holds information used by UltraLite during query processing. In particular, it holds temporary tables that store intermediate results for queries that involve ORDER BY or GROUP BY clauses.

Tip

You can safely delete the temporary file without loss of data—as long as UltraLite is not running. It does not contain information that may be required across sessions.

UltraLite data and state management

UltraLite maintains state information that includes:

- ◆ Synchronization progress counts, to ensure that synchronization occurs successfully.
- ◆ Row state, to maintain data integrity by tracking how data has changed between synchronizations.
- ◆ Indexing, to access the row data efficiently.
- ◆ 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

- ◆ [“The UltraLite database schema” on page 15](#)

- ◆ [“Built-in synchronization features for UltraLite”](#) [*MobiLink - Client Administration*]

Understanding concurrency in UltraLite

UltraLite databases may receive multiple concurrent requests. To design applications that handle concurrent requests properly, you should understand how UltraLite manages concurrency in the database.

It is helpful to separate several concepts when thinking about concurrent database access. These concepts are ordered from high-level to low-level:

- ◆ **Applications** The UltraLite engine can respond to requests from multiple separate applications. Other versions of the UltraLite runtime permit only a single application to connect to a database at a time.

☞ For more information, see [“Choosing a data management component”](#) on page 11.

- ◆ **Threads** UltraLite supports multi-threaded applications. A single application may be written to use multiple threads, each of which may connect to the database.
- ◆ **Connections** Even a single-threaded application can open multiple connections to a database. In any case, individual connections can employ only a single thread.
- ◆ **Transactions** Each connection can have a single transaction in progress at any one time. Transactions can consist of a single request or multiple requests. Data modifications made during a transaction are not permanent in the database until the transaction is committed. Either all data modifications made in a transaction are committed, or all are rolled back.
- ◆ **Requests** A transaction consists of one or more requests. A request may be a query that reads data, or an insert, update, or delete that modifies data, or a synchronization.
- ◆ **The current row** When an application is working with the result set of a query, UltraLite maintains a pointer to the **current row** within the result set. In some interfaces, this current row is tracked explicitly using a cursor (a pointer to a position in a result set). In others, the application uses methods on a result set object or table object to identify and change the current row. Such methods use a cursor in the application's code.

Multiple databases

The UltraLite runtime can manage a maximum of four databases at any one time. A single UltraLite application can open multiple connections to separate databases. No concurrency issues arise from such applications, as the data in each database is independent.

Locking

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. An attempt to change a locked row sets error `SQLCODE SQLE_LOCKED`, while an attempt to change a deleted row sets the error `SQLCODE_SQL_NOTFOUND`. Your applications should check the `SQLCODE` value after attempting to modify data.

Re-reading rows

UltraLite operates at an isolation level of 0. Consequently, your applications may experience **dirty reads**—that is, they may access rows in the database that are not committed and consequently may still get rolled back by another transaction.

For example, consider two connections, A and B, each with their own transaction.

As connection A works with the result set of a query, UltraLite **fetches** a copy of the current row into a buffer. If 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. At that time, a write lock is placed on the row to prevent other transactions from modifying it. The change to the database is not permanent until connection A commits the transaction.

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.

If connection B does modify the current row, the row becomes locked. In this case, connection A cannot modify the current row. If connection A fetches the current row again, and the row has been deleted, connection A gets the next row in the result set. If the row has been modified, connection A gets the latest copy of the row. If the columns of the index used by connection A have been modified, connection A sees the change as a delete followed by an insert, and so gets the next row in the result set.

Synchronization

Synchronization behaves as a separate connection. 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 access to data during synchronization by setting the Disable Concurrency synchronization parameter.

Tip

If synchronization fails, UltraLite supports resumable downloads on all platforms. See “[Handling failed downloads](#)” [*MobiLink - Server Administration*].

 For more information, see “[UltraLite Clients](#)” [*MobiLink - Client Administration*] and “[Disable Concurrency synchronization parameter](#)” [*MobiLink - Client Administration*].

How UltraLite tracks row states

Maintaining row state information is a powerful part of the UltraLite feature set. Without this built-in functionality, you would have to implement logic to keep track of the current position in open tables. Tracking the state of tables and rows is particularly important for data synchronization.

Each row in an UltraLite database has a one-byte marker to keep track of the state of the row. The row states are used to control transaction processing, recovery, and synchronization. Different actions produce different results with commits and rollbacks:

- ◆ **When a delete is issued** The state of each affected row is changed to reflect the fact that it was deleted. Rolling back a delete is as simple as restoring the original state of the row.
- ◆ **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 until the next synchronization confirms the delete with the consolidated database. 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 an update is synchronized, both the old and new versions of the row are needed to allow conflict detection and resolution.

All old row versions are deleted after synchronization. If a row is updated many times between synchronizations, only the oldest version of the row and the most recent version of the row are kept.

Transaction processing, recovery, and backup

UltraLite provides transaction processing. A transaction is a logical set of operations that are executed atomically; that is, either all operations in the transaction are stored in the database or none are. If a transaction is **committed**, all operations are stored in the database; if a transaction is **rolled back**, none are.

Some transactions consist of a single operation. Many programming interfaces use an **autocommit** setting to commit a transaction after each operation. 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.

Recovery from system failure

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

UltraLite does not use a transaction log to perform recovery. Instead, UltraLite uses the state byte for every row to determine the fate of a row when recovering. When a row is inserted, updated, or deleted in an UltraLite database, the state of the row is modified to reflect the operation and the connection that performed the operations. 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 on recovery.

☞ For information about state bytes, see [“How UltraLite tracks row states” on page 18](#).

Backups

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

However, you can also just copy the database file to a desktop computer as a manual backup mechanism in smaller UltraLite deployments.

Indexes in UltraLite databases

UltraLite supports the notion of database indexes. Indexes can greatly improve the performance of searches on the indexed column(s). However, indexes take up space within the database file and slow down insert, update and delete operations.

When UltraLite optimizes a SQL query, it checks to see if indexes exist that can help return the results more efficiently.

You can index single or multiple column types—with the exception of LONG VARCHAR and LONG BINARY columns. Every table in an UltraLite database must have at least one index: the primary key.

Tip

You can use the UltraLite table API to iterate over the rows of an index in a predictable order.

See also

- ◆ [“Working with UltraLite indexes” on page 65](#)
- ◆ [“Index performance considerations” on page 33](#)
- ◆ [“max_hash_size property” on page 106](#)
- ◆ [“Database page size considerations” on page 36](#)
- ◆ [“page_size property” on page 112](#)

Part II. Using UltraLite Databases

This part describes how to create and maintain UltraLite databases.

CHAPTER 2

Creating and Configuring UltraLite Databases

Contents

Creating UltraLite databases 24
Choosing creation-time database properties 30
Configuring post-creation database options 40

About this chapter

This chapter describes how to create and configure UltraLite databases. It discusses the methods of creation you can use and lists the considerations for the database properties and options you can set.

Creating UltraLite databases

Irrespective of the database creation method you employ, you must always configure the database properties that control and/or define characteristics of the database when you create a database. You do not need to configure any database properties if you are comfortable with the default values. However, once the database has been created, you cannot re-configure these values. If you need to change any database property, you must re-create the database.

You can categorize database creation methods into two categories:

- ◆ Desktop creation methods with UltraLite administration tools designed for this purpose. This chapter will primarily focus on these methods.
- ◆ On-device creation methods with UltraLite APIs. On-device creation methods are primarily described in each API specific UltraLite programming guide. These methods are briefly described here for reasons of completeness only.

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

Sharing a database among multiple platforms

Within the configuration differences imposed by different operating systems, you might be able to copy the database from one device to another. If you are unsure of property compatibility among multiple platforms, you should create a database in Sybase Central with the Create Database wizard for UltraLite. This wizard handles the file compatibility logic for you, which prevents you from creating a file that is not supported on your combination of deployment devices.

See also

- ◆ [“Choosing creation-time database properties” on page 30](#)
- ◆ [“The UltraLite database file” on page 15](#)
- ◆ [“UltraLite Connection String Parameters Reference” on page 127](#)
- ◆ [“Viewing UltraLite database settings” on page 78](#)
- ◆ [“Working with UltraLite Databases” on page 55](#)

Desktop creation

The UltraLite Create Database wizard Choose this method if you need help navigating the available database creation properties. 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 then record and subsequently use with the `ulcreate` utility. See [“Creating an UltraLite database from Sybase Central” on page 25](#).

The MobiLink Create Synchronization Model wizard Choose this method, if:

- ◆ You are creating a complex synchronization system with multiple remote UltraLite databases and a centralized consolidated database.
- ◆ You need to use reference database other than a SQL Anywhere database.

See [“Creating an UltraLite database from a MobiLink synchronization model”](#) on page 26.

The command line Choose any of the following utilities:

- ◆ Use the `ulcreate` utility if you are very familiar with database creation properties and want a fast alternative to creating a new, but empty, UltraLite database. This option is particularly useful for creating databases in batch operations. See [“Creating an UltraLite database from the command prompt”](#) on page 25.
- ◆ Use the `ulinit` utility if you want to create a new, but empty, UltraLite database sourced from a SQL Anywhere reference database schema. See [“Creating an UltraLite database from a SQL Anywhere reference database”](#) on page 26.
- ◆ 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 XML”](#) on page 28.

Creating an UltraLite database from Sybase Central

You can create a database in Sybase Central using the UltraLite Create Database wizard.

◆ To create a new UltraLite database (Sybase Central)

1. Start Sybase Central by choosing Start ► Programs ► SQL Anywhere 10 ► Sybase Central.
2. Create an UltraLite database by choosing Tools ► UltraLite ► Create Database.
The Create Database wizard appears.
3. Follow the instructions in the wizard.

See also

- ◆ [“Creating an UltraLite database from the command prompt”](#) on page 25
- ◆ [“Creating an UltraLite database from a SQL Anywhere reference database”](#) on page 26
- ◆ [“Creating an UltraLite database from XML”](#) on page 28
- ◆ [“Choosing creation-time database properties”](#) on page 30
- ◆ [“Upgrading UltraLite”](#) [*SQL Anywhere 10 - Changes and Upgrading*]

Creating an UltraLite database from the command prompt

You can create a database from a command prompt with the `ulcreate` utility. With this utility, you can include utility options to specify different properties for the database.

◆ To create a new UltraLite database (command prompt)

1. Open a command prompt.
2. Run the `ulcreate` utility, including any necessary parameters.

For example, to create a case-sensitive UTF-8 database called *test.udb*, such that it overwrites a database if it already exists, run the command with the following syntax:

```
ulcreate -c "DBF=test.udb" -o "case=respect:utf_encoding=1" -y
```

Notice that supplying a database file in a connection string is simply an alternative to specifying the database file after the other command prompt options. For example:

```
ulcreate -o "case=respect:utf_encoding=1" -y test.udb
```

See also

- ◆ [“Creating an UltraLite database from Sybase Central” on page 25](#)
- ◆ [“Creating an UltraLite database from a SQL Anywhere reference database” on page 26](#)
- ◆ [“Creating an UltraLite database from XML” on page 28](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“Choosing creation-time database properties” on page 30](#)
- ◆ [“Upgrading UltraLite” \[*SQL Anywhere 10 - Changes and Upgrading*\]](#)

Creating an UltraLite database from a MobiLink synchronization model

MobiLink includes a modeling system that allows you to create a remote client with schema definitions and table maps created with the Create Synchronization Model wizard. This wizard creates a file that contains the information about the synchronization model (remote and consolidated databases).

Once you have created your model, you can work in Model mode in Sybase Central and 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 UltraLite remote.

See [“Creating and configuring a model” \[*MobiLink - Getting Started*\]](#).

Creating an UltraLite database from a SQL Anywhere reference database

A reference database is a SQL Anywhere database that serves as a template for the UltraLite database you want to create. Your UltraLite database is a subset of the columns, tables, and indexes in this reference database. You select these objects as part of a publication in the reference database.

In previous versions of UltraLite, a reference database was required by some modes of development. Now, unless you are using the SQL Anywhere database as a consolidated database to collect and manage data from multiple remote databases, there is no clear development advantage to using this method—unless you want to first model your data with an architecture tool like Sybase's PowerDesigner Physical Data Model.

You can initialize an UltraLite database with the ulinit utility for UltraLite. With this utility, you can include utility options to specify different properties for the database.

◆ To initialize/extract a new UltraLite database from a reference database (command prompt)

1. Create a new SQL Anywhere database.

You can either create a new one with the dbinit utility or use Sybase Central. You can also create a SQL Anywhere database from non-SQL Anywhere databases, by migrating data from these third-party files.

Set database properties with UltraLite usage in mind

The UltraLite database is generated with the same property settings as those in the reference database. By setting these options in the reference database, you also control the behavior of your UltraLite database. These options include:

- ◆ Date format
- ◆ Date order
- ◆ Nearest century
- ◆ Precision
- ◆ Scale
- ◆ Time format
- ◆ Timestamp format

2. Prepare the reference database by adding objects required by the UltraLite database.

- ◆ **Tables and keys** Add the tables and remember to set primary keys as they are required by UltraLite. If you need to, you can also assign foreign key relationships that you need within your UltraLite application. You can use any convenient tool, such as Sybase Central or Sybase PowerDesigner Physical Data Model, or another database design tool.
- ◆ **Indexes**
If your UltraLite applications frequently retrieve information in a particular order, consider adding an index to your reference database specifically for this purpose. An index can improve performance dramatically, particularly on slow devices. Note that primary keys are automatically indexed, but other columns are not.
- ◆ **Publications** If you want to synchronize different tables at different times, you can do so using multiple UltraLite specific publications to define table subsets and set synchronization priority with them.

For information on multiple MobiLink server synchronization options, see “[Publications in UltraLite](#)” [*MobiLink - Client Administration*].

3. Run the ulinit utility, including any necessary options.

For example, to initialize an UltraLite database called *customer.udb* with tables contained in two distinct publications, enter the following command line at a command prompt. Specifically, Pub1 may contain a small subset of tables for priority synchronization, while Pub2 could contain the bulk of the data.

```
ulinit -a DBF=MySource.db  
-c DBF=customer.udb -n Pub1 -n Pub2
```

See also

- ◆ “Creating an UltraLite database from Sybase Central” on page 25
- ◆ “Creating an UltraLite database from the command prompt” on page 25
- ◆ “Creating an UltraLite database from XML” on page 28
- ◆ “UltraLite Initialize Database utility (ulinit)” on page 172
- ◆ “Choosing creation-time database properties” on page 30
- ◆ “Upgrading UltraLite” [*SQL Anywhere 10 - Changes and Upgrading*]

Creating an UltraLite database from XML

You can use XML as an intermediate format for managing your UltraLite database, provided that the format follows the requirements for UltraLite usage. You can use XML as follows:

- ◆ 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 that you can check into a version control system.

UltraLite cannot use an arbitrary XML file. The `<install-dir>\win32` directory contains a `usm.xsd` file. Use this file to review the XML format.

◆ To create an UltraLite database from an XML file

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. See “Exporting relational data as XML” [*SQL Anywhere Server - SQL Usage*].
 - ◆ 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. However, always ensure that the format of the XML meets the UltraLite requirements.
2. Open a command prompt.
3. Run the `ulload` utility, including any necessary parameters.

For example, to create a new UltraLite database in the file `sample.udb` from the table formats and data in `sample.xml`:

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

See also

- ◆ “Creating an UltraLite database from Sybase Central” on page 25
- ◆ “Creating an UltraLite database from the command prompt” on page 26
- ◆ “Creating an UltraLite database from a SQL Anywhere reference database” on page 26
- ◆ “Upgrading UltraLite” [*SQL Anywhere 10 - Changes and Upgrading*]
- ◆ “UltraLite Load XML to Database utility (ulload)” on page 174
- ◆ “Choosing creation-time database properties” on page 30

On-device creation

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. Populating the database is then simply a question synchronizing with a consolidated database.

Considerations

This option is not as ideal as other available database creation options: by adding all the additional database creation and SQL code, your application size can grow considerably. Nonetheless, this option can simplify deployment because you only need to deploy the application to the device. Additionally, in some pre-production development cycles, you may want to delete and reconstruct the database on your device for testing purposes.

See also

- ◆ [“Desktop creation” on page 24](#)
- ◆ UltraLite for C/C++: [“CreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite.NET: [“CreateDatabase method” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite for AppForge: [“CreateDatabase method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method createDatabase” \[UltraLite - M-Business Anywhere Programming\]](#)

Choosing creation-time database properties

UltraLite database creation properties, written to the database via all administration tools or even the CreateDatabase function, are recorded as *name=value* pairs. UltraLite stores these properties in system tables, which means that users and/or applications can access them in the same way. See “[sysuldata system table](#)” on page 197.

Accessing properties

You cannot change properties after you have created a database. However, you can view them in Sybase Central. See “[Viewing UltraLite database settings](#)” on page 78.

You can also access them programmatically from the UltraLite application. Typically, applications should not access the data in these tables directly. Instead, your applications should call the GetDatabaseProperty function appropriate to the API.

For API specific details, see:

- ◆ UltraLite C/C++: “[GetDatabaseProperty function](#)” [*UltraLite - C and C++ Programming*]
- ◆ MobileVB: “[GetDatabaseProperty method](#)” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “[GetDatabaseProperty method](#)” [*UltraLite - .NET Programming*]
- ◆ M-Business: “[Method getDatabaseProperty](#)” [*UltraLite - M-Business Anywhere Programming*]

In addition to these properties, you can further configure other aspects of your database with either database options or connection parameters.

☞ For database options you can set at any time, see “[Configuring post-creation database options](#)” on page 40. For parameters you configure at connection time, see “[Connecting to an UltraLite Database](#)” on page 43 and “[UltraLite Connection String Parameters Reference](#)” on page 127.

Property list	Description
case	Sets the case-sensitivity of string comparisons in the UltraLite database. See “ Case sensitivity considerations ” on page 33 and “ case property ” on page 94.
checksum_level	Sets the level of checksum validation in the database. See “ Verifying page integrity with checksums ” on page 37 and “ checksum_level property ” on page 95.
date_format	Sets the default string format in which dates are retrieved from the database. See “ Date considerations ” on page 34 and “ date_format property ” on page 97.
date_order	Controls the interpretation of date ordering of months, days, and years. See “ Date considerations ” on page 34 and “ date_order property ” on page 100.
fips	Controls AES FIPS compliant data encryption using a Certicom certified cryptographic algorithm. See “ Security considerations ” on page 38 and “ fips property ” on page 102.

Property list	Description
max_hash_size	Set the maximum number of bytes that are used to hash the UltraLite indexes. See “Index performance considerations” on page 33 and “max_hash_size property” on page 106 .
nearest_century	Controls the interpretation of two-digit years in string-to-date conversions. See “Nearest century conversion considerations” on page 35 and “nearest_century property” on page 109 .
obfuscate	Controls whether or not to obfuscate data in the database. Obfuscation is a form of simple encryption. See “Security considerations” on page 38 and “obfuscate property” on page 110 .
page_size	Defines the database page size. See “Database page size considerations” on page 36 and “page_size property” on page 112 .
precision	Specifies the maximum number of digits in the result of any decimal arithmetic. See “Decimal point position considerations” on page 36 and “precision property” on page 114 .
scale	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision. See “Decimal point position considerations” on page 36 and “scale property” on page 116 .
time_format	Sets the format for times retrieved from the database. See “Time considerations” on page 34 and “time_format property” on page 118 .
timestamp_format	Determines how the timestamp is formatted in UltraLite. See “Timestamp considerations” on page 35 and “timestamp_format property” on page 120 .
timestamp_increment	Determines how the timestamp is truncated in UltraLite. See “Timestamp considerations” on page 35 and “timestamp_increment property” on page 123 .
utf8_encoding	Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode. See “Character considerations” on page 31 and “utf8_encoding property” on page 125 .

Character considerations

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.

You cannot change the collation of the database once UltraLite writes the file. To change the collation that an existing database uses, do the following: unload the database, create a new database using the appropriate collation, and then reload the database. You may need to convert the data to match the new collation.

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

- ◆ The sort order you require. Generally speaking, you should 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 UTF-8 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 UTF-8 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.

See also

- ◆ [“Platform requirements for character set encoding” on page 32](#)
- ◆ [“utf8_encoding property” on page 125](#)
- ◆ [“Overview of character sets, encodings, and collations” \[*SQL Anywhere Server - Database Administration*\]](#)
- ◆ [“UltraLite Connection String Parameters Reference” on page 127](#)
- ◆ [“Character Set Considerations” \[*MobiLink - Server Administration*\]](#)
- ◆ [“Case sensitivity considerations” on page 33](#)
- ◆ [“Security considerations” on page 38](#)

Platform requirements for character set encoding

Each platform has specific character set and encoding requirements.

Palm OS

Never use UTF-8 encoding; Palm does not support Unicode characters. Always choose a collation which matches the code page of your intended device.

Symbian OS

Always use UTF-8 encoding. This is the natural character set of the device. Symbian does not support the MBCS character sets.

Windows desktop and Windows CE

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

- ◆ [“utf8_encoding property” on page 125](#)
- ◆ [“Overview of character sets, encodings, and collations” \[SQL Anywhere Server - Database Administration\]](#)
- ◆ [“UltraLite Connection String Parameters Reference” on page 127](#)
- ◆ [“Character Set Considerations” \[MobiLink - Server Administration\]](#)
- ◆ [“Case sensitivity considerations” on page 33](#)
- ◆ [“Security considerations” on page 38](#)

Index performance considerations

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 consequently how much of the actual value UltraLite hashes, is determined by the hash size you set.

UltraLite databases automatically use a default maximum hash size of 4 bytes. You can change this default to another size, or change it to 0 to disable index hashing. You can override this database default hash size when you create a new index.

How a hash improves performance

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 (that is, 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. Consequently 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.

How you determine this database default hash size requires that you evaluate the tradeoff between query efficiency and database size: the higher the maximum hash value, the larger the database size grows.

See also

- ◆ [“Working with UltraLite indexes” on page 65](#)
- ◆ [“Index performance considerations” on page 33](#)
- ◆ [“max_hash_size property” on page 106](#)
- ◆ [“Database page size considerations” on page 36](#)

Case sensitivity considerations

The results of comparisons on strings, and the sort order of strings, in part depends on the case sensitivity of the database. Case sensitivity in UltraLite databases affects:

- ◆ **Data** The case sensitivity of the data is reflected in indexes and in string comparisons. By default, comparisons are always case insensitive. For example, UltraLite compares **short sleeve t** and **Short Sleeve T** equally.

- ◆ **Identifiers** Identifiers include table names, column names, and so on. Identifiers, including user IDs, are always case insensitive, regardless of the database case sensitivity. For example, UltraLite always compares **UID=DBA** and **UID=dba** equally.
- ◆ **Passwords** Passwords are always case sensitive in UltraLite databases. For example, UltraLite does not compare **PWD=sql** and **PWD=sQl** equally.

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.

See also

- ◆ [“Character considerations” on page 31](#)
- ◆ [“case property” on page 94](#)

Date considerations

Values whose data types are DATE are represented in a format set by the `date_format` property. Date values can, however, also be represented by strings. Before it can be retrieved, a date value must be assigned to a string variable.

Date parts you can use

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, the second (and parts thereof).

Use the `date_order` property to arrange these date parts in a specific order.

ISO (that is, 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.

See also

- ◆ [“date_order property” on page 100](#)
- ◆ [“date_format property” on page 97](#)
- ◆ [“Timestamp considerations” on page 35](#)

Time considerations

UltraLite writes times from time parts you set with the `time_format` property. Time parts can include hours, minutes, seconds (and milliseconds).

Time values can, however, also be represented by strings. Before it can be retrieved, a time value must be assigned to a string variable.

If you are not using the default ISO format (that is, HH:MM:SS), you must specify the format of these time parts.

ISO (that is, 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.

See also

- ◆ [“time_format property” on page 118](#)
- ◆ [“Timestamp considerations” on page 35](#)

Timestamp considerations

UltraLite creates a timestamp from date and time parts that you set with the `date_format` and `time_format` properties. Together, date and time total seven parts (that is, year, month, day, hour, minute, second, and millisecond).

Timestamp values can, however, also be represented by strings. Before it can be retrieved, a timestamp value must be assigned to a string variable.

Typically timestamp columns ensure data integrity is maintained when synchronizing with a consolidated database. Time stamps 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 maintain timestamps and timestamp increments to the same resolution. By setting these properties to match that of the consolidated database, you can help to avoid spurious inequalities.

See also

- ◆ [“Date considerations” on page 34](#)
- ◆ [“Time considerations” on page 34](#)
- ◆ [“timestamp_increment property” on page 123](#)
- ◆ [“timestamp_format property” on page 120](#)
- ◆ [“Timestamp-based downloads” \[*MobiLink - Server Administration*\]](#)
- ◆ [“Understanding concurrency in UltraLite” on page 17](#)

Nearest century conversion considerations

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. In the case of a two-digit date, you need to set the appropriate rollover value.

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.

However, you should always store four-digit dates to avoid issues with incorrect conversions and keep dates unambiguous. See [“Using unambiguous dates and times” \[SQL Anywhere Server - SQL Reference\]](#).

- ◆ **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.

If this option is not set, the default setting of 50 is assumed. Thus, two-digit year strings are understood to refer to years between 1950 and 2049.

See also

- ◆ [“nearest_century property” on page 109](#)
- ◆ [“Ambiguous string to date conversions” \[SQL Anywhere Server - SQL Reference\]](#)

Decimal point position considerations

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 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.

See also

- ◆ [“precision property” on page 114](#)
- ◆ [“scale property” on page 116](#)

Database page size considerations

The unit of storage within a database is known as a page, and all database input and output operations are always carried out a page at a time. UltraLite allocates a page to hold either:

- ◆ Table rows (user or system tables)
- ◆ Index information
- ◆ MobiLink server synchronization information

If you are encrypting your database, UltraLite encrypts the data at the page level.

Choosing an optimal page size

UltraLite databases are stored in pages, and all I/O operations are carried out a page at a time. For most applications, the UltraLite default page size of 4 KB is appropriate.

However, you can choose another size if your database deployment requires it. Just remember that the page size you choose can affect the performance or size of the database. The typical rule of thumb is that larger databases usually benefit from a larger page size, because smaller pages hold less information and may force less efficient use of space—particularly if you insert rows that are slightly more than half a page in size:

- ◆ **Number of rows** The larger the database, the more row that can be stored on each page. Because a row (excluding BLOBs) must fit on a page, the page size determines how large the largest row can be. In general, smaller page sizes are likely to benefit operations that retrieve a relatively small number of rows from random locations. By contrast, larger pages tend to benefit queries that perform sequential table scans. In this situation, reading one page into memory to obtain the values of one row may have the side effect of loading the contents of the next few rows into memory.
- ◆ **Cache size** Large page sizes require larger cache sizes, because fewer large pages can fit into the same space. Should you choose a larger page size, such as 8 KB, you may want to increase the size of the cache when you connect to the database.

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. How many pages is enough depends entirely on your database and the nature of the queries your application performs. You can conduct performance tests with various cache sizes. If your cache cannot hold enough pages, performance suffers as UltraLite begins swapping frequently-used pages to disk. See [“CACHE_SIZE connection parameter” on page 128](#).

- ◆ **Index entries** Page size also affects indexes. The larger the database page, the more index entries it can hold. See [“Working with UltraLite indexes” on page 65](#).

However, there are instances when a smaller page size is in order. For example, small page sizes allow UltraLite to run with fewer resources because it can store more pages in a cache of the same size. Small pages are particularly useful if your database must run on small devices with limited memory. They can also help in situations when you use your database primarily to retrieve small pieces of information from random locations.

Verifying page integrity with checksums

You can check the validity of the pages stored to disk, flash, or memory by setting the `checksum_level` database property. Depending on the level you choose, UltraLite calculates then records a **checksum** for each database page, before it writes the page to storage. See [“checksum_level property” on page 95](#).

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 verification fails,

UltraLite stops the database and reports fatal error. This error cannot be corrected; you must re-create your UltraLite database and report the database failure to iAnywhere.

See also

- ◆ [“Indexes in UltraLite databases” on page 20](#)
- ◆ [“page_size property” on page 112](#)
- ◆ [“Connecting to an UltraLite Database” on page 43](#)

Security considerations

By default, UltraLite databases are unencrypted on disk. Text and binary columns are plainly readable within the database when using a viewing tool such as a hex editor. If you need to encrypt data for greater security, consider the options listed below.

- ◆ **Obfuscation** This option provides protection against casual attempts to access data in the database. It does not provide as much security as strong encryption. Obfuscation has minimal performance impact. You set obfuscation with the `obfuscate` property. End users do not need to supply a corresponding connection parameter.

For details on how to use the `obfuscate` property, see [“obfuscate property” on page 110](#).

- ◆ **AES 128-bit strong encryption** UltraLite databases can be strongly encrypted using the AES 128-bit algorithm, which is the same algorithm used to encrypt SQL Anywhere databases. Use of strong encryption provides security against skilled and determined attempts to gain access to the data, but has a significant performance impact. You set encryption in the wizards by selecting the `Encrypt Database` option and then selecting `AES Strong Encryption`. Using a creation utility, you set the key with the `key` connection parameter. This same parameter is used by end users when connecting to the database after it has been created. See [“DBKEY connection parameter” on page 143](#).
- ◆ **AES FIPS 140-2 compliant encryption** UltraLite provides encryption libraries compliant with the FIPS 140-2 US and Canadian government standard (using a Certicom certified cryptographic module). Only choose this option if you are a government agency that requires this strength of encryption. You set FIPS compliant encryption with the `FIPS` property. End users need to then supply the required key in the corresponding connection parameter.

For more information, see [“fips property” on page 102](#) and [“DBKEY connection parameter” on page 143](#).

Tip

The MobiLink server's synchronization streams can use public/private keys to encrypt streamed data. For ease of deployment, you can embed these certificates in the UltraLite database when you create it. For details, see [“Configuring MobiLink clients to use transport-layer security” \[SQL Anywhere Server - Database Administration\]](#).

Notes

Both the FIPS and AES database encryption types use 128-bit AES. This means that if you use the same encryption key, the database is encrypted the same way irrespective of the standard you choose.

Caution

You can change the encryption key after the database has been created, but only under extreme caution. See any of the following:

- ◆ UltraLite for C++: “[ChangeEncryptionKey function](#)” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for AppForge: “[ChangeEncryptionKey method](#)” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “[ChangeEncryptionKey method](#)” [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: “[Method changeEncryptionKey](#)” [*UltraLite - M-Business Anywhere Programming*]

This operation is costly and is non-recoverable: if your operation terminates mid-course, you will lose your database entirely.

Additionally, if you lose or forget the encryption key for a strongly encrypted database, there is no way to access the data in it—even with the assistance of technical support. The database must be discarded and you must create a new database.

See also

- ◆ “[fips property](#)” on page 102
- ◆ “[obfuscate property](#)” on page 110
- ◆ “[DBKEY connection parameter](#)” on page 143
- ◆ UltraLite for AppForge: “[Encryption and obfuscation](#)” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “[Encryption and obfuscation](#)” [*UltraLite - .NET Programming*]
- ◆ UltraLite for C++: “[Encrypting data](#)” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for M-Business Anywhere: “[Database encryption and obfuscation](#)” [*UltraLite - M-Business Anywhere Programming*]

Configuring post-creation database options

There are two database options you can set in UltraLite databases, both of which identify the database during MobiLink server synchronization.

Remote ID considerations for MobiLink server synchronization

The **remote ID** is a unique identifier employed by an UltraLite remote that is used for MobiLink synchronization. The MobiLink remote ID is initially set to NULL and the MobiLink server sets it to a GUID during a database's first synchronization (or if you reset the remote ID to NULL again). However, the remote ID can be any string that has meaning to you, provided that the string remains unique among all remote MobiLink clients. The uniqueness requirement is always enforced.

You use the remote ID to store the synchronization progress for the MobiLink user name. By including a unique remote ID, user names are no longer required to be unique. The user name can now be a true user name that is used to identify a client for authentication.

The remote ID becomes particularly useful when you have multiple MobiLink users synchronizing the same UltraLite client database. In this case, your synchronization scripts should reference the remote ID and not just the user name.

◆ To set/reset a remote ID in Sybase Central

1. In the Folders view, right-click the UltraLite database you have connected to, and choose Options from the popup menu.

The Database Options dialog appears.

2. Select the ml_remote_id table entry.
3. In the Value field, type the new value for the ID.
4. Click Set Now to save your changes.
5. Click Close.

See also

- ◆ “Introducing UltraLite as a MobiLink client” [[MobiLink - Client Administration](#)]
- ◆ “Remote IDs” [[MobiLink - Client Administration](#)]
- ◆ “UltraLite user authentication” [[MobiLink - Client Administration](#)]
- ◆ “User Name synchronization parameter” [[MobiLink - Client Administration](#)] and “Password synchronization parameter” [[MobiLink - Client Administration](#)]
- ◆ “ml_remote_id option” on page 108

Global database ID considerations

The global ID sets a starting value for GLOBAL AUTOINCREMENT columns in order to maintain primary key uniqueness when synchronizing with the MobiLink server. 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. For more details, see [“Using global autoincrement” \[MobiLink - Server Administration\]](#).

◆ **To set/reset a global database ID in Sybase Central**

1. In the Folders view, right-click the UltraLite database you have connected to, and choose Options from the popup menu.

The Database Options dialog appears.

2. Select the `global_database_id` table entry.
3. In the Value field, type the new value for the ID.
4. Click Set Now to save your changes.
5. Click Close.

See also

- ◆ [“global_id option” on page 104](#)

CHAPTER 3

Connecting to an UltraLite Database

Contents

Introducing UltraLite database connections 44
Opening connections with connection strings 47
Storing parameters with the ULSQLCONNECT environment variable 53

About this chapter

This chapter describes how UltraLite administration tools and UltraLite applications connect to UltraLite databases. It introduces various connection methods you can employ such as connection strings or the ULSQLCONNECT parameter.

Introducing UltraLite database connections

Any application that uses a database must establish a connection to that database before any transactions can occur. An application can be an UltraLite command line utility, a connection dialog from either Sybase Central tool or Interactive SQL, 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.

List of UltraLite database connection parameters

Some of the connection parameters you configure overlap with properties you define at creation time. At creation time, you typically are defining the property required. At connection time, you must supply the value that was configured at creation time. It is important that you understand the differences and commonalities of these shared settings so you know what is expected and when.

Parameter name	Description
CACHE_SIZE	Defines the size of the database cache. See “CACHE_SIZE connection parameter” on page 128 .
CON	Specifies a name of the current connection. See “CON connection parameter” on page 130 .
DBF, and CE_FILE, PALM_FILE, NT_FILE, or SYMBIAN_FILE	<p>At creation-time these parameters set the location of the database. For subsequent connections, they tell UltraLite where to find the file.</p> <p>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.</p> <p>See:</p> <ul style="list-style-type: none"> ◆ “DBF connection parameter” on page 131 ◆ “CE_FILE connection parameter” on page 133 ◆ “PALM_FILE connection parameter” on page 137 ◆ “NT_FILE connection parameter” on page 135 ◆ “SYMBIAN_FILE connection parameter” on page 140
PALM_DB	AppForge development only. Sets the correct UltraLite creator ID so that creator ID is used, rather than the AppForge creator ID. Use this with the PALM_FILE parameter. See “PALM_DB connection parameter” on page 139
DBN	Identifies a running database by name rather than file name. See “DBN connection parameter” on page 142 .
DBKEY	At creation-time, this parameter sets the encryption key to use. For subsequent connections, names and then passes the same encryption key for the database. If the incorrect key is named, the connection fails. See “DBKEY connection parameter” on page 143 .

Parameter name	Description
PALM_ALLOW_BACKUP	Controls backup behavior over HotSync on Palm devices. See “PALM_ALLOW_BACKUP connection parameter” on page 145 .
PWD	At creation-time, sets the <i>initial</i> password for a user. For subsequent connections, supplies the password for the user ID. See “PWD connection parameter” on page 146 .
RESERVE_SIZE	Pre-allocates the file system space required for your UltraLite database without actually inserting any data. See “RESERVE_SIZE connection parameter” on page 148 .
START	Specifies the location of the UltraLite engine executable and then starts it. See “START connection parameter” on page 149 .
UID	At creation-time, sets the <i>initial</i> user ID. For subsequent connections, identifies a user to the database. The user ID must be one of up to four user IDs stored in the UltraLite database. See “UID connection parameter” on page 150 .

See also

- ◆ [“Interpreting user ID and password combinations” on page 46](#)
- ◆ [“page_size property” on page 112](#)

The role of user authentication

In UltraLite, you cannot disable authentication. A successful connection requires that a user be authenticated. Unlike SQL Anywhere, UltraLite database users are created and managed solely for the purposes of authentication—and not for the purposes of object ownership. Once a user authenticates and connects to the database, the user has unrestricted access to everything in that database—including schema data.

You can only add or modify UltraLite users from an existing connection. Therefore, any changes to your UltraLite user-base can only occur after you have connected with a valid UID and PWD.

If this is your first time connecting, the UID and PWD required are the same values set when you first created the database. If you did not set an initial user, then you must authenticate with the defaults of **UID=DBA** and **PWD=sql**.

Bypassing authentication

While you may not be able to disable authentication, you can bypass it simply by using UltraLite defaults both when you create and connect to the database.

By not supplying both the UID *and* the PWD parameters—irrespective of the connection method you use—UltraLite will then always assume the defaults of **UID=DBA** and **PWD=sql**.

◆ To bypass authentication in UltraLite

1. Do not set a UID and PWD parameters when you create a database.

2. Do not delete or modify the default user in your UltraLite database.
3. Do not set a UID and PWD parameters when you connect to the database you have created.

See also

- ◆ [“Considerations and limitations” on page 76](#)
- ◆ [“Working with UltraLite users” on page 76](#)
- ◆ [“Interpreting user ID and password combinations” on page 46](#)

Interpreting user ID and password combinations

UltraLite allows you to set one, none, or both of the UID and PWD parameters—except when a partial definition prevents a user from being identified by UltraLite. The table below tells you how UltraLite interprets incomplete user definitions.

If you create a database with...	It has this impact...
No user ID <i>and</i> password.	UltraLite creates a default user with a UID of DBA and PWD of sql . You do not need to supply these parameter upon future connection attempts.
The user ID parameter only. Examples: <ul style="list-style-type: none">◆ UID=JaneD◆ UID=JaneD;PWD=◆ UID=JaneD;PWD=""	UltraLite creates a default user with a UID of JaneD and an empty PWD. When connecting, you must always supply the UID parameter. The PWD parameter is not required.
The password parameter only. Examples: <ul style="list-style-type: none">◆ PWD=3saBys◆ UID=;PWD=3saBys◆ UID="";PWD=3saBys	UltraLite generates an error. UltraLite cannot set a password without a user ID.

See also

- ◆ [“The role of user authentication” on page 45](#)
- ◆ [“Considerations and limitations” on page 76](#)
- ◆ [“Working with UltraLite users” on page 76](#)

Opening connections with connection strings

A connection string is an assembled set of parameters that is passed from an application to the runtime, so that a connection can be defined and ultimately established.

There are three steps that take place before a connection to a database is opened:

1. **The parameter definition phase** You must define the connection via any combination of supported parameters. Some connection parameters are always required to open a connection. Others are used to adjust database features for a single connection.

How these parameters are supplied can vary depending on whether you are connecting from an UltraLite administration tool, or an UltraLite application. See [“Supplying UltraLite connection parameters” on page 47](#) for details.

2. **The string assembly phase** Either you or the application assembles the supplied parameters into a string. Connection strings contain a list of parameters defined as *keyword=value* pairs in a semi-colon delimited list. For example, a connection string fragment that supplies a file name, user ID, and password is written as follows:

```
DBF=myULdb.ldb;UID=JDoe;PWD=token
```

See [“Assembling parameters into connection strings” on page 49](#) for details.

3. **The transmittal phase** When the connection string has been assembled, it is passed to the database via an UltraLite API to the UltraLite runtime for processing. If the connection attempt is validated, the connection is granted. Connection failures can occur if:

- ◆ The database file supplied does not exist.
- ◆ Authentication was unsuccessful.

Supplying UltraLite connection parameters

How connection information is collected depends on how systematic or automated you require the input to be. The more systematic the input, the more reliable the connection information is.

Connection details can be collected via different methods, depending on whether or not you are connecting from a custom UltraLite application, or any of the SQL Anywhere administration tools for UltraLite (that is, Interactive SQL, UltraLite command line utilities, and UltraLite wizards in Sybase Central).

Method	Administration tools	Custom applications
<p><i>Prompt the end user at connection time:</i> when you require a user to authenticate as one of the four supported database users. The UltraLite graphical administration tools use a connection object.</p> <p>Where possible, use either the ULConnectionParms or ConnectionParms object. It provides easier checking and a more systematic interface than using a connection string that is an argument for the Open method.</p> <p>For details, see any of the following:</p> <ul style="list-style-type: none"> ◆ UltraLite for AppForge: “Authenticating users” [UltraLite - AppForge Programming] ◆ UltraLite.NET: “Authenticating users” [UltraLite - .NET Programming] ◆ UltraLite C/C++ : “Authenticating users” [UltraLite - C and C++ Programming] ◆ UltraLite for M-Business Anywhere: “Authenticating users” [UltraLite - M-Business Anywhere Programming] ◆ UltraLite for embedded SQL: “Authenticating users” [UltraLite - C and C++ Programming] 	X	X
<p><i>Use a connection string:</i>if user authentication is not required because the deployment is to a single-user device, or it is too awkward to prompt a user each time they start the application. The UltraLite command line utilities typically use a connection string if a connection to a database is required. You can also program your UltraLite application to read the values from a stored file, or hard code it into your application.</p> <p>For details, see any of the following:</p> <ul style="list-style-type: none"> ◆ UltraLite for AppForge: “Connecting to an UltraLite database” [UltraLite - AppForge Programming] ◆ UltraLite.NET: “Connecting to a database” [UltraLite - .NET Programming] ◆ UltraLite C/C++ : “Connecting to a database” [UltraLite - C and C++ Programming] ◆ UltraLite for M-Business Anywhere: “Connecting to an UltraLite database” [UltraLite - M-Business Anywhere Programming] ◆ UltraLite for embedded SQL: “Connecting to a database” [UltraLite - C and C++ Programming] 	X ¹	X ³

Method	Administration tools	Custom applications
<p><i>Use the ULSQLCONNECT environment variable:</i> if you want to store connection parameters you use repeatedly, so you don't need to repeatedly provide them while your UltraLite database is under development on the desktop. Values you supply as a parameter in ULSQLCONNECT become defaults for the UltraLite desktop tools.</p> <p>All UltraLite desktop administration tools check the ULSQLCONNECT values for any missing parameters not supplied in a connection string, following parameter precedence rules. To override these values, simply supply the alternate value in the connection string.</p> <p>For details on precedence is evaluated in UltraLite, see “Precedence of parameters for UltraLite administration tools” on page 49. For details on how to set the ULSQLCONNECT environment variable, see “Storing parameters with the ULSQLCONNECT environment variable” on page 53.</p>	X ²	N/A

1 Typically user-supplied.

2 For desktop administration tools only.

3 Typically hard-coded or stored in a file.

Assembling parameters into connection strings

An assembly of connection parameters supplied in any application's connection code (be it an administration tool or a custom UltraLite application) is called a connection string. In some cases, applications parse the fields of a ConnectionParms object into a string. In others however, you 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 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 named database file until you supply this string:

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

Notes

UltraLite does not ignore unrecognized connection string parameters. Instead, it generates an error for unrecognized connection parameters.

Precedence of parameters for UltraLite administration tools

All of the UltraLite administration tools follow a specific order of parameter precedence:

- ◆ If specified, CE_FILE, NT_FILE, SYMBIAN_FILE, and PALM_FILE parameters always take precedence over DBF.
- ◆ If you use two DBF parameters concurrently, the last one specified takes precedence.
- ◆ 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 ULSQLCONNECT or a connection object.
- ◆ ULSQLCONNECT environment variable is then checked to supply parameters that are not supplied in the connection string.
- ◆ If no value for *both* UID and PWD is supplied in either the connection string or ULSQLCONNECT, the defaults of **UID=DBA** and **PWD=sql** are assumed.

Limitations

Any leading spaces and/or trailing spaces in connection string parameter values are ignored. All connection string parameters cannot include leading single quotes ('), leading double quotes ("), or semi-colons (;).

See also

- ◆ [“Storing parameters with the ULSQLCONNECT environment variable” on page 53](#)

Specifying file paths in a connection parameter

The physical storage of your device determines whether or not the database is saved as a file, and what naming conventions you must follow when identifying your database with one of the supported connection parameters (that is, DBF, CE_FILE, NT_FILE, PALM_FILE, or SYMBIAN_FILE).

The DBF parameter is most appropriate when targeting a single deployment platform or when using UltraLite desktop administration tools. For example, utility usage examples in this book will most often use the DBF parameter:

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

Windows CE Tip

You can use the UltraLite administration tools to administer databases already deployed to an attached device. For details, see [“Windows CE” on page 51](#).

Otherwise, if you are writing a cross-platform application, you should use the platform-specific (CE_FILE, NT_FILE, PALM_FILE, or SYMBIAN_FILE) file parameters to construct a universal connection string. For example, if you are developing an AppForge component for Windows CE and Palm OS, your connection string might look as follows:

```
Connection = DatabaseMgr.OpenConnection("UID=JDoe;PWD=ULdb;  
CE_FILE=\\database\\MyCEDB.udb;PALM_FILE=MyPalmDB")
```

Windows desktop

Windows desktops do not have much in the way of file name restrictions. Desktops allow either absolute or relative paths.

Windows CE

Windows CE devices require that all paths be absolute.

You can administer a CE file on either the desktop or on the attached device. To identify a file on a CE device, ensure you prefix the absolute path with **wce:**. For example, using the unload utility:

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

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

Windows CE and backups

It should be noted that if you are using the Unload Database and Upgrade Database wizards to administer a database on the attached CE device, UltraLite cannot back up the database before the unload or action occurs. You must perform this action manually before running these wizards.

Palm OS

Palm OS does not necessarily use the concept of file paths. Therefore, how you define it depends on the store type (that is, record-based, or VFS).

File-based stores (VFS) For databases on a VFS volume, define the file with the following syntax:

```
vfs: [ volume-label: | volume-ordinal: ] filename
```

You can set the *volume-label* as **INTERNAL** for the built-in drive, or **CARD** for either an expansion card or the label name of the volume. There is no default string for the *volume-label*.

Alternatively, you can set the *volume-ordinal* to identify the volume. Since the enumeration of mounted volumes can vary, ensure you set the correct ordinal volume for your chosen internal or external volume. The default value is 0 (which is the first volume enumerated by the platform).

For the *filename*, always specify the absolute file path, following the file and path naming convention of Palm OS. If directories specified in the path do not already exist, they are created.

Record-based data stores For record-based data stores, database names must follow all conventions for Palm OS database names (for example, database names cannot exceed the 32 character limit and cannot contain a path).

Also ensure you use the appropriate value for DBF or PALM_FILE according to the database's location.

- ◆ Use the *.PDB* extension with DBF when you store a Palm OS database anywhere other than on the device itself (for example, with ulload).

- ◆ Once you move the file to the device, the *.PDB* extension is dropped by the HotSync conduit. For example, if the database you created on the desktop is named *CustDB.pdb*, then when you deploy it to the device, the filename changes to *CustDB*.

Symbian OS

On Symbian, paths must be absolute. Otherwise, Symbian OS assumes the file is in the device's root directory.

The database is stored in the file system, similar to that used by Windows desktop and Windows CE devices. For example Symbian phones may use the *C:* drive for phone memory, and perhaps the *D:* or *E:* drive for flash cards. Phone RAM may use the *Z:* drive. Letter designations can vary among the models used.


See also

- ◆ [“DBF connection parameter” on page 131](#)
- ◆ [“NT_FILE connection parameter” on page 135](#)
- ◆ [“CE_FILE connection parameter” on page 133](#)
- ◆ [“PALM_FILE connection parameter” on page 137](#)
- ◆ [“SYMBIAN_FILE connection parameter” on page 140](#)

Storing parameters with the ULSQLCONNECT environment variable

The ULSQLCONNECT environment variable is optional, and is not set by the installation program. ULSQLCONNECT contains a list of parameters defined as *keyword=value* pairs in a semi-colon delimited list.

Use ULSQLCONNECT to avoid having to repeatedly supply the same connection parameters during development to UltraLite administration tools. You cannot use ULSQLCONNECT for custom applications.

 For a list of supported connection parameters in UltraLite, see [“Supplying UltraLite connection parameters” on page 47](#).

Caution

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

◆ To set ULSQLCONNECT for UltraLite desktop tools

1. At the command prompt, enter the following:

```
set ULSQLCONNECT="parameter=value; ..."
```

2. If an administration tool requires any additional parameter 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 variable. See [“Precedence of parameters for UltraLite administration tools” on page 49](#) for details.

Example

To use ULSQLCONNECT to connect to a file named *c:\database\myfile.udb* connecting as a user called **demo** and a password of **test**, set the following variable in your environment:

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

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

For example, if you were using `ulload` to add additional information to your database from an *extra.xml* file, you would simply run the command as follows:

```
ulload -a extra.xml
```

See also

- ◆ [“Precedence of parameters for UltraLite administration tools” on page 49](#)

CHAPTER 4

Working with UltraLite Databases

Contents

Working with UltraLite tables and columns 56

Working with UltraLite indexes 65

Working with UltraLite publications 72

Working with UltraLite users 76

Viewing UltraLite database settings 78

About this chapter

This chapter describes the mechanics of creating, altering, and deleting database objects such as users, tables, indexes, and publications.

Working with 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 or even open databases.

See also

- ◆ [“Database tables” \[SQL Anywhere 10 - Introduction\]](#)
- ◆ [“Designing Your Database” \[SQL Anywhere Server - SQL Usage\]](#)
- ◆ [“UltraLite system tables” on page 192](#)

Creating UltraLite tables

You can create new tables to hold your relational data, either with SQL statements in Interactive SQL or with Sybase Central.

In UltraLite, you can only create base tables, which you declare in order to hold persistent data. The table and its data continue to exist until you explicitly delete the data or drop the table. UltraLite does not support global temporary or declared temporary tables.

Note

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.

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected database.

◆ To create an UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database contents, open the Tables folder.
3. From the File menu, choose New ► Table.
The Table Creation wizard appears.
4. In the Table Creation wizard, enter a name for the new table.
5. Click Finish.

6. On the Columns tab in the right pane, add columns to the table.
7. Choose File ► Save Table when finished.

Interactive SQL

In Interactive SQL, you can declare columns while creating a new table.

◆ To create an UltraLite table (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE TABLE statement.

Example The following statement creates a new table to describe the various skills and professional aptitudes of employees within a company. The table has columns to hold an identifying number, a name, and a type (for example, technical or administrative) for each skill.

```
CREATE TABLE Skills (  
    SkillID INTEGER PRIMARY KEY,  
    SkillName CHAR( 20 ) NOT NULL,  
    SkillType CHAR( 20 ) NOT NULL  
)
```

See also

- ◆ [“CREATE TABLE statement” on page 331](#)
- ◆ [“Adding a column to an UltraLite table” on page 58](#)

Using allsync and nosync suffixes

You can append either **_allsync** or **_nosync** to a table name to control data restriction for synchronization. You can use these suffixes as an alternative to using publications to control data restrictions. To control data priority, define one or more publications.

- ◆ If you create a table with a name ending in **_allsync**, all rows of that table are synchronized at each synchronization—even if they have not changed since the last synchronization.

Tip

You can store user-specific or client-specific data in allsync tables. If you upload the data in the table to a temporary table in the consolidated database on synchronization, you can use the data to control synchronization by your other scripts without having to maintain that data in the consolidated database.

- ◆ If you create a table with a name ending in **_nosync**, all rows of that table are excluded from synchronization. You can use these tables for persistent data that is not required in the consolidated database's table.

Example

In the *CustDB.udb* sample database, you can see one table was declared a nosync table because the table name is defined as **ULIdentifyEmployee_nosync**. Therefore, no matter how data changes in this table, it is never synchronized with MobiLink and information will not appear in the *CustDB.db* consolidated database.

See also

- ◆ [“Working with UltraLite publications” on page 72](#)
- ◆ [“Designing synchronization in UltraLite” \[MobiLink - Client Administration\]](#)
- ◆ [“Nosync tables in UltraLite” \[MobiLink - Client Administration\]](#)
- ◆ [“Allsync tables in UltraLite” \[MobiLink - Client Administration\]](#)
- ◆ [“Exploring the CustDB Samples for UltraLite” on page 79](#)

Adding a column to an UltraLite table

You can add a new column easily if the table is empty. However, if the table already holds data, you can only add a column if the column definition includes a default value or allows NULL values.

You can use either Sybase Central or directly execute a SQL statement (for example, Interactive SQL) to perform this task.

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected table.

◆ To add a new column to an UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. Open the Tables folder.
3. In the Columns tab, right-click the background and select New ► Column.
4. Set all new attributes for the column. You must declare a name, a data type, constraints, whether or not the table allows null values, and so on.
5. Choose File ► Save Table when finished.

Interactive SQL

In Interactive SQL, you can only declare columns while creating or altering a table.

◆ To add columns to a new UltraLite table (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE TABLE statement or ALTER TABLE, ensuring that you define columns by declaring the name, and other attributes accordingly.

Examples The following example 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.

```
CREATE TABLE borrowed_book (
    loaner_name CHAR(100) PRIMARY KEY,
    date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
    date_returned DATE,
    book CHAR(20)
)
```

The following example modifies the customer table to now include a column for addresses that can hold up to 50 characters:

```
ALTER TABLE customer
ADD address CHAR(50)
```

See also

- ◆ “Choosing column names” [*SQL Anywhere Server - SQL Usage*]
- ◆ “Data types in UltraLite” on page 212
- ◆ “Choosing data types for columns” [*SQL Anywhere Server - SQL Usage*]
- ◆ “CREATE TABLE statement” on page 331
- ◆ “ALTER TABLE statement” on page 322

Altering UltraLite column definitions

You can change the structure of column definitions for a table by altering various column attributes, or even deleting columns entirely. 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.

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

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected table.

◆ To alter an existing UltraLite column (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database contents, open the Tables folder.
3. In the Columns tab, make the necessary attribute changes.
4. Choose File ► Save Table when finished.

Interactive SQL

In Interactive SQL, you can perform these tasks with the ALTER TABLE statement.

◆ **To alter an existing UltraLite column (Interactive SQL)**

1. Connect to the UltraLite database.
2. Execute an ALTER TABLE statement.

Make changes carefully

The following examples show how to change the structure of the database. In all these cases, the statement is immediately committed. So, once you make the change, any item referring to this table may no longer work.

Examples The following statement shortens the SkillDescription column from a maximum of 254 characters to a maximum of 80:

```
ALTER TABLE Skills
MODIFY SkillDescription CHAR( 80 )
```

The following statement deletes the Classification column:

```
ALTER TABLE Skills
DROP Classification
```

The following statement changes the name of the entire table:

```
ALTER TABLE Skills
RENAME Qualification
```

See also

- ◆ “Choosing column names” [[SQL Anywhere Server - SQL Usage](#)]
- ◆ “Data types in UltraLite” on page 212
- ◆ “Choosing data types for columns” [[SQL Anywhere Server - SQL Usage](#)]
- ◆ “ALTER TABLE statement” on page 322

Dropping UltraLite tables

Dropping a table means that you are deleting it from the database. You can drop any table provided that 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.

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

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected table.

◆ To delete an UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database contents, open the Tables folder for that database.
3. Select the table and then choose Edit ► Delete.

Interactive SQL

In Interactive SQL, deleting a table is also called dropping it. You can drop a table by executing a DROP TABLE statement.

◆ To delete an UltraLite table (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a DROP TABLE statement.

Examples The following DROP TABLE command deletes all the records in the Skills table and then removes the definition of the Skills table from the database:

```
DROP TABLE Skills
```

Like the CREATE statement, the DROP statement automatically executes a COMMIT statement before and after dropping the table. This makes all changes to the database since the last COMMIT or ROLLBACK permanent. The DROP statement also drops all indexes on the table.

See also

- ◆ [“DROP TABLE statement” on page 339](#)

Browsing the information in UltraLite tables

You can use Sybase Central or Interactive SQL to browse the data held within the tables of an UltraLite database. 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 doesn't have a notion of ownership, all users can browse all tables.

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected database.

◆ To browse UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. If system tables are hidden and you want to browse the data in one or more tables, right-click the background of the Contents pane and choose Show System Objects.
3. To display tables, click Tables.

4. To browse the data of a Table, double-click a table name.
5. Click the Data tab in the right pane.

◆ To filter UltraLite system tables (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database contents, right-click the database name you are connected to and click either Hide System Objects or Show System Objects.

Interactive SQL

In Interactive SQL, you can perform these tasks with the SELECT statement.

◆ To browse UltraLite user tables (Interactive SQL)

1. Connect to a database.
2. Execute a SELECT statement, specifying the user table you want to browse.

◆ To browse UltraLite system tables (Interactive SQL)

1. Connect to a database.
2. Execute a SELECT statement, by the system table you want to browse.

Example For example, to display the contents of the table systable on the Results tab in the Results pane in Interactive SQL, execute the following command:

```
SELECT *  
FROM SYSTABLE
```

See also

- ◆ [“UltraLite System Table Reference” on page 191](#)

Copying and pasting data to/from UltraLite databases

With Sybase Central, you have different copying and pasting as well as dragging and dropping options for tables or columns. This 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.

Target	Result
Another UltraLite/SQL Anywhere database.	A new object is created, and the original object's code is copied to the new object.
The same UltraLite database.	A copy of the object is created; you must rename the new object.

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

In Sybase Central, when you copy any of the objects from the list that follows, the SQL for the object is copied to the clipboard so it can be pasted 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.

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

Interactive SQL

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

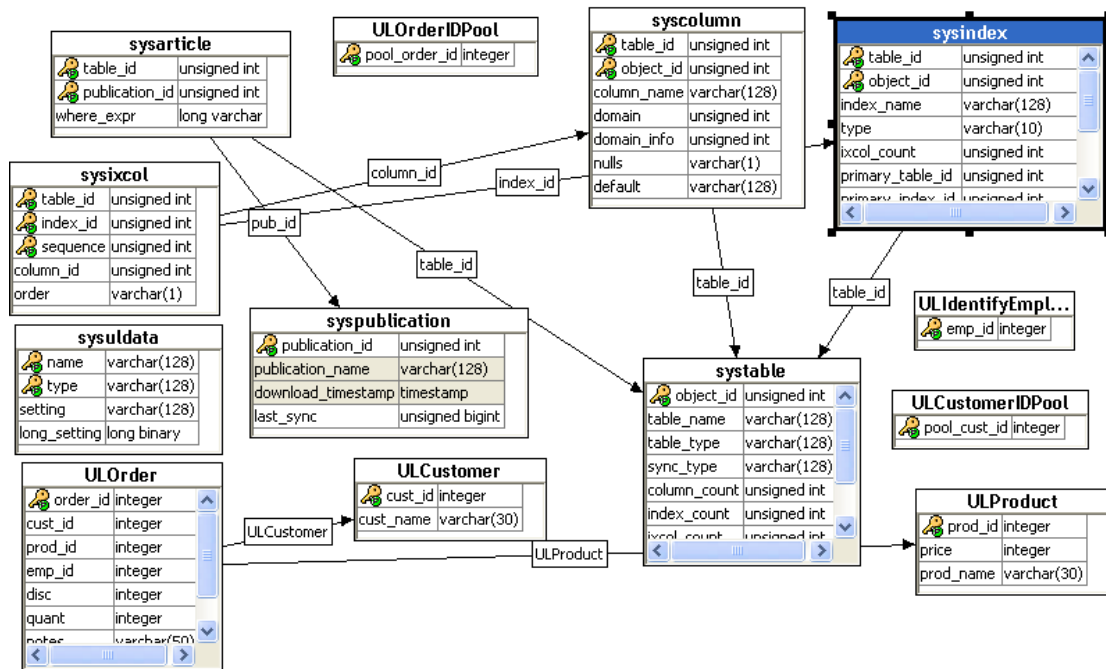
- ◆ Use the SELECT statement results into a named object.
- ◆ Use the INSERT statement to insert a row or selection of rows from elsewhere in the database into a table.

See also

- ◆ [“Copying database objects in the SQL Anywhere plug-in” \[SQL Anywhere Server - Database Administration\]](#)
- ◆ [“INSERT statement” on page 340](#)
- ◆ [“SELECT statement” on page 342](#)

Viewing 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 design concepts” [[SQL Anywhere Server - SQL Usage](#)]
- ◆ “Entity-relationship diagrams” [[SQL Anywhere Server - SQL Usage](#)]

Working with UltraLite indexes

An index provides an ordering (either ascending or descending) of a table's rows based on the values in one or more columns. 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 or not a query is using a particular index, you can check the query access plan with Interactive SQL. Alternatively, your UltraLite applications can include PreparedStatement objects which have a method to return plans.

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

- ◆ [“Query access plans in UltraLite” on page 232](#)
- ◆ [“Composite indexes” \[SQL Anywhere Server - SQL Usage\]](#)
- ◆ UltraLite for AppForge: [“Working with data using the table API” \[UltraLite - AppForge Programming\]](#).
- ◆ UltraLite for AppForge: [“ULPreparedStatement class” \[UltraLite - AppForge Programming\]](#).
- ◆ UltraLite.NET: [“Accessing and manipulating data with the Table API” \[UltraLite - .NET Programming\]](#).
- ◆ UltraLite.NET: [“Prepare method” \[UltraLite - .NET Programming\]](#).
- ◆ UltraLite for C++: [“Accessing data with the Table API” \[UltraLite - C and C++ Programming\]](#).
- ◆ UltraLite for C++: [“UltraLite_PreparedStatement class” \[UltraLite - C and C++ Programming\]](#).
- ◆ UltraLite for M-Business Anywhere: [“Working with data using the table API” \[UltraLite - M-Business Anywhere Programming\]](#).
- ◆ UltraLite for M-Business Anywhere: [“Class PreparedStatement” \[UltraLite - M-Business Anywhere Programming\]](#).

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 and 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. This is because 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. Perhaps simply maintaining UltraLite default indexes (indexes for primary keys and for unique constraints) may be sufficient.
- ◆ **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.

Choosing an index type

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.

Index characteristic	Unique keys	Unique indexes	Non-unique indexes
Allows duplicate index entries for rows that have the same values in indexed columns.	no	no	yes
Allows null values in index columns.	no	yes	yes

Notes

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 UltraLite indexes” on page 67](#)

Adding UltraLite indexes

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

Note

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

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected database.

◆ To create a new index for a given UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database contents, open the Indexes folder.
3. From the File menu, choose New ► Index.

The Index Creation wizard appears.

4. Name the index and select the table from the list. Click Next.
5. Follow the instructions in the wizard.

New indexes appear in the Indexes folder.

Interactive SQL

In Interactive SQL, you can perform these tasks with the CREATE INDEX statement.

◆ To create a new index for a given UltraLite table (Interactive SQL)

1. Connect to an UltraLite database.
2. Execute a CREATE INDEX statement. See [“CREATE INDEX statement” on page 328](#).

This 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.

Examples 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
```

See also

- ◆ [“CREATE INDEX statement” on page 328](#)

Dropping an index

Dropping an index deletes it from the database.

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

Sybase Central

In Sybase Central, you can perform these tasks while working with a selected database.

- ◆ **To drop an UltraLite index (Sybase Central)**

1. Connect to the UltraLite database.
2. While browsing the database contents, open the Indexes folder.
3. Select the desired index and then choose Edit ► Delete.

Interactive SQL

In Interactive SQL, deleting a table is also called dropping it. You can perform these tasks with the DROP INDEX statement.

- ◆ **To drop an UltraLite index (Interactive SQL)**

1. Connect to a database.
2. Execute a DROP INDEX statement.

Example The following statement removes the EmployeeNames index from the database:

```
DROP INDEX EmployeeNames
```

See also

- ◆ [“DROP INDEX statement” on page 337](#)

Tuning performance with index hashing

You can tune the performance of your queries by choosing a specific size for the **hash**. The hash appends a suffix or key to the original index entry. The key provides an index optimization, because it aims to avoid the expensive operation of finding, loading, and then unpacking the rows to determine the indexed value.

How the hash key enhances query performance

A row ID allows UltraLite to locate the row for the actual data in the table. A row ID is always part of a hashed index entry. If you set the hash size to 0 (that is, 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. Consequently 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 values from the row.

For example, if you had a table named `PRODUCT_LIST`, you may write a query to return all rows that hold the value of **Apples** in this table.

```
SELECT *
FROM PRODUCT_LIST
WHERE Items = 'Apples'
```

If the index for the `Items` column only contains row IDs, UltraLite uses that row ID to find and unpack the data, before it can compare that value with **Apples**. However, if you hashed your index so that it included the first four bytes with the row ID, the index entry might appear as **riD(75) Blue**. In this case UltraLite does not locate, unpack, nor compare the actual row value; UltraLite already knows that this row does not satisfy the `WHERE` clause in the example, without having to locate, unpack and compare the row value of "blueberries".

On the other hand, another index entry might appear as **riD(32)Appl**. In this case, UltraLite does need to locate, unpack, and compare the value of "Apple pie", before it can determine that this row also does not satisfy the `WHERE` clause.

See also

- ◆ [“Choosing an optimal hash size” on page 69](#)
- ◆ [“Indexes in UltraLite databases” on page 20](#)
- ◆ [“Adding UltraLite indexes” on page 67](#)

Choosing an optimal hash size

The UltraLite default maximum hash size of 4 bytes was carefully chosen to suit most deployments. You can increase the size to include more data with the row ID; however, you will increase the size of the index and consequently the size of the database as a result.

When choosing a hash size, consider the data type, the row data, and the database size as described in the following sections. Because you need to consider all three conditions, choosing an optimal hash size can be a very complex task. Especially when you consider the fact that a bigger hash size does not always translate into improved database performance due to increased storage requirements—and how different operating systems handle those increases.

The only way to determine if you have chosen an optimal hash size, is to run performance tests against your UltraLite client application on the target device. You need to observe how various hash sizes affect the application and query performance in addition to the changes in database size itself.

The data type

Different data types require a different maximum hash size. If you want to hash the entire value in a column, note the maximum required by each data type in the table that follows. However, you may not need to hash the entire value, depending on the requirements of your implementation.

Data type	Bytes used to hash the entire value
FLOAT, DOUBLE, and REAL	not hashed
BIT and TINYINT	1
SMALL INT and SHORT	2
INTEGER, LONG and DATE	4
DATETIME, TIME, TIMESTAMP, and BIG	8
CHAR and VARCHAR	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.
BINARY	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.
UUID	16

For example, if you set a maximum hash size of 6 bytes, and if you had a two-column composite index that you declared as INTEGER and BINARY (20) respectively, then based on the data type size requirements, the following occurs:

- ◆ The entire value of the row in the INTEGER column is hashed and stored in the index. Only 4 bytes are required to hash integer data types.
- ◆ Only the first 2 bytes of the BINARY column is hashed and stored in the index, because this is the first 4 bytes are used by the INTEGER column. If 6 bytes do not hash an appropriate amount of the BINARY column, increase the maximum hash size, so more of the BINARY column is hashed.

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 may render the hash ineffective if you choose a size that only hashes prefixes. In this case you need to choose a size that hashes the unique values as well as the common prefixes—or even choose to not hash the index.

The database size

Each page has some fixed overhead, but the majority of the page space is used by the actual index entries. Therefore, an index hash uses some of the database page space that might otherwise be used to store more entries. The fewer entries that can fit on the page, the larger the database becomes. Use a smaller maximum hash size if you require a compact database.

For example, depending on whether you require a small database or improved index performance, you could make arguments for either a larger or a smaller hash size. Consider the tables listed in the table that follows.

Table	Page Size	Hash Size	Number of Entries	Pages required
Table A	4 KB	0	1200	3 pages
Table B	4 KB	32 bytes	116	3 pages
Table C	4 KB	32 bytes	1200 entries	11 pages

See also

- ◆ [“Indexes in UltraLite databases” on page 20](#)
- ◆ [“Adding UltraLite indexes” on page 67](#)
- ◆ [“Data types in UltraLite” on page 212](#)

Setting the hash size

You can set hash size in two ways:

- ◆ To store a database *default* for the maximum size, you can set a database property called `max_hash_size` 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.
- ◆ If you want to *override* the default set with this parameter, you can set a specific hash size when you create a new index. You can do this in two ways:
 - ◆ 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

- ◆ [“CREATE TABLE statement” on page 331](#)
- ◆ [“CREATE INDEX statement” on page 328](#)
- ◆ [“max_hash_size property” on page 106](#)

Working with UltraLite publications

A publication is a database object that identifies a subset of UltraLite data you want to synchronize with the MobiLink server at one time. If you wish 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 selected rows from a table. You can define this set of rows with a WHERE clause (except with HotSync on Palm OS).

Each database can have multiple publications, depending on the synchronization logic you require. 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.
- ◆ The publication determines which columns are selected, but it does not determine the order in which they are sent. 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” \[MobiLink - Client Administration\]](#)
- ◆ [“Publishing data” \[MobiLink - Client Administration\]](#)
- ◆ [“Designing synchronization in UltraLite” \[MobiLink - Client Administration\]](#)
- ◆ [“Introduction to synchronization scripts” \[MobiLink - Server Administration\]](#)

Publishing whole UltraLite tables

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

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

Sybase Central

In Sybase Central, you can perform this task while working with the connected database.

◆ To publish one or more whole UltraLite tables (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database contents, open the Publications folder.
3. Create the publication.
Choose File ► New ► Publication. The Create Publication wizard appears.
4. Enter a name for the new publication, and click Next.
5. On the Tables tab, select a table from the list of Available Tables. Click Add. The table appears in the list of Selected Tables on the right.
6. Optionally, you may add additional tables. The order of the tables is not important.
7. Click Finish.

Interactive SQL

In Interactive SQL, you can perform this task with the CREATE PUBLICATION statement.

◆ To publish one or more whole UltraLite tables (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE PUBLICATION statement that specifies the name of the new publication and the table you want to publish.

Example The following statement creates a publication that publishes the whole customer table:

```
CREATE PUBLICATION pub_customer (  
    TABLE customer  
)
```

See also

- ◆ [“CREATE PUBLICATION statement” on page 330](#)
- ◆ [“UltraLite Clients” \[MobiLink - Client Administration\]](#)

Publishing a subset of rows from an UltraLite table

You can create a publication that contains specific rows of a table from Sybase Central, or by specifying a WHERE clause in the CREATE PUBLICATION statement (except with HotSync on Palm OS). Both techniques rely on the WHERE clause to limit the rows to be uploaded to those that have changed *and* that satisfy a search condition in the WHERE clause.

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

Tip

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

What you cannot use in a WHERE clause

The search condition in the WHERE clause can only reference columns that are included in the article. In addition, you cannot use any of the following in the WHERE clause:

- ◆ subqueries
- ◆ variables
- ◆ non-deterministic functions

These conditions are not enforced but breaking them can lead to unexpected results. Any errors relating to the WHERE clause are generated at runtime, and not when the publication is defined.

Sybase Central

In Sybase Central, you can perform this task while working with the connected database.

◆ To publish only some rows in an UltraLite table (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database contents, open the Publications folder.
3. Create a new publication.
Choose File ► New ► Publication. The Create Publication wizard appears.
4. Enter a name for the new publication. Click Next.
5. On the Tables tab, select a table from the list of Available Tables. Click Add. The table is added to the list of Selected Tables on the right.
6. On the WHERE Clauses tab, select the table and enter the search condition in the lower box. Optionally, you can use the Insert dialog to assist you in formatting the search condition.
7. Click Finish.

Interactive SQL

In Interactive SQL, you can perform this task with the CREATE PUBLICATION statement.

◆ To create a publication in UltraLite using a WHERE clause (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a CREATE PUBLICATION statement that includes the tables you want to include in the publication and a WHERE condition.

Example The following example creates a single-article publication that includes all sales order information for sales rep number 856:

```
CREATE PUBLICATION pub_orders_samuel_singer
( TABLE SalesOrders
  WHERE SalesRepresentative = 856 )
```

See also

- ◆ [“CREATE PUBLICATION statement” on page 330](#)
- ◆ [“UltraLite Clients” \[MobiLink - Client Administration\]](#)

Dropping a publication for UltraLite

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

Sybase Central

In Sybase Central, you can perform this task while working with the connected database.

◆ To drop a publication (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database contents, open the Publications folder.
3. Right-click the desired publications and choose Delete from the popup menu.

Interactive SQL

In Interactive SQL, deleting a publication is also called dropping it. You can perform this task with the DROP PUBLICATION statement.

◆ To drop a publication (Interactive SQL)

1. Connect to the UltraLite database.
2. Execute a DROP PUBLICATION statement.

Example The following statement drops the publication named pub_orders:

```
DROP PUBLICATION pub_orders
```

See also

- ◆ [“DROP PUBLICATION statement” on page 338](#)
- ◆ [“UltraLite Clients” \[MobiLink - Client Administration\]](#)

Working with UltraLite users

Because user IDs and passwords are encrypted in the UltraLite database, you can only view the list of defined users in Sybase Central.

Considerations and limitations

When creating unique user IDs, bear the following limitations in mind:

- ◆ UltraLite user IDs are separate from MobiLink user names and from other SQL Anywhere user IDs.
- ◆ UltraLite supports up to four unique users.
- ◆ Both the user ID and password values have a limit of 31 characters.
- ◆ Passwords are always case sensitive and user IDs are always case insensitive. You can change a password anytime from Sybase Central.
- ◆ Any leading or trailing spaces the user ID are ignored. The user ID cannot include leading single quotes ('), leading double quotes ("), or semi-colons(;).
- ◆ As a precaution, UltraLite hashes the password before saving it. Therefore, you can only modify the password in Sybase Central.
- ◆ You cannot change a user ID once it is created. Instead, you must delete the user ID in question and then add a new one.

Adding a new UltraLite user

UltraLite does not support the creation of users with Interactive SQL. However, you can add users by:

- ◆ Using Sybase Central to add users to the User folder. See the procedure below for details.
- ◆ Using the GrantConnectTo function on the Connection object to add new users from an UltraLite application.

◆ To create a new UltraLite user (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database contents, open the Users folder.
3. From the File menu, choose New ► User.

The User Creation wizard appears.

4. Follow the instructions in the wizard. Ensure you understand how UltraLite interprets different user ID and password combinations. See [“Interpreting user ID and password combinations”](#) on page 46 for details.

See also

- ◆ UltraLite for AppForge: “Authenticating users” [[UltraLite - AppForge Programming](#)]
- ◆ UltraLite.NET: “Authenticating users” [[UltraLite - .NET Programming](#)]
- ◆ UltraLite C/C++ : “Authenticating users” [[UltraLite - C and C++ Programming](#)]
- ◆ UltraLite for M-Business Anywhere: “Authenticating users” [[UltraLite - M-Business Anywhere Programming](#)]
- ◆ UltraLite for embedded SQL: “Authenticating users” [[UltraLite - C and C++ Programming](#)]

Deleting an existing UltraLite user

UltraLite does not support the deletion of users with Interactive SQL. However, you can delete users by:

- ◆ Using Sybase Central to delete users from the User folder. See the procedure below for details.
- ◆ Using the RevokeConnectFrom function on the Connection object to remove users from an UltraLite application.

◆ To delete an existing UltraLite user (Sybase Central)

1. Connect to the UltraLite database.
2. While browsing the database, click Users.
3. Right-click the user in the Users pane and click Delete.

See also

- ◆ UltraLite for AppForge: “Authenticating users” [[UltraLite - AppForge Programming](#)]
- ◆ UltraLite.NET: “Authenticating users” [[UltraLite - .NET Programming](#)]
- ◆ UltraLite C/C++ : “Authenticating users” [[UltraLite - C and C++ Programming](#)]
- ◆ UltraLite for M-Business Anywhere: “Authenticating users” [[UltraLite - M-Business Anywhere Programming](#)]
- ◆ UltraLite for embedded SQL: “Authenticating users” [[UltraLite - C and C++ Programming](#)]

Viewing UltraLite database settings

You can only view configured database properties in Sybase Central. You cannot change them after the database is created. You can, however, change database options (that is, `global_ID` and `ml_remote_ID`) at any time.

◆ To browse UltraLite database properties (Sybase Central)

1. Connect to the database.
2. While browsing connected databases, right-click a database and select Properties.

The Database Properties dialog appears.

3. Select the Extended Information tab.

Database properties are listed alphabetically by the property name. To sort database properties by the value, click the Value column.

Note

You cannot modify UltraLite database properties after you have created the database. If you need to change a property, you need to create a new database and then load the data into it.

4. If you think database properties have changed since you started browsing them, click Refresh.

◆ To browse or modify UltraLite database options (Sybase Central)

1. Connect to the database.
2. While browsing connected databases, right-click a database and select Options.

The Database Options dialog box appears.

3. If you want to set or reset an option, type a new value in the Value field.
4. Click either Set Now or Reset Now to commit the change.

See also

- ◆ [“Creating UltraLite databases” on page 24](#)
- ◆ [“UltraLite Database Settings Reference” on page 93](#)

CHAPTER 5

Exploring the CustDB Samples for UltraLite

Contents

Introducing CustDB	80
Finding CustDB sample files	81
Lesson 1: Logging in and populating the UltraLite remote	83
Lesson 2: Using the CustDB client application	84
Lesson 3: Synchronizing with the CustDB consolidated database	86
Lesson 4: Browsing MobiLink synchronization scripts	88
What's next?	90

About this chapter

CustDB (Customer Database) helps you learn about various aspects of UltraLite as part of a multi-tiered database management solution that includes MobiLink synchronization with a SQL Anywhere consolidated database. The desktop-based tutorials in this chapter use CustDB examples to demonstrate features and behaviors of UltraLite.

Introducing CustDB

What is CustDB?

CustDB is used as the example for tutorials in MobiLink and UltraLite development guides to help users of the software explore features in a guided and instructional manner. CustDB is a simple example of point-of-sale customer management solution that consists of the following:

- ◆ A **consolidated** SQL Anywhere database. This data is pre-populated with sales-status data.
- ◆ A **remote** UltraLite database. This database is initially empty.
- ◆ An UltraLite client application.
- ◆ A MobiLink server synchronization sample with synchronization scripts already created for you.

While different versions of the application code exists for each supported programming interface and platform, subsequent tutorials reference the compiled version of the application for Windows desktops only. It is important to remember that each version implements UltraLite features with some variation to conform to the conventions of each platform.

What does it do?

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.

☞ For more information on the CustDB usage scenario, see “[Scenario](#)” [*MobiLink - Getting Started*]. For more details on the users in the CustDB sample and the types of actions they can perform, see “[Users in the CustDB sample](#)” [*MobiLink - Getting Started*].

Goals of UltraLite CustDB tutorials

After following each lessons you will know 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 command line utilities.

See also

- ◆ “[Finding CustDB sample files](#)” on page 81
- ◆ “[Tables in the CustDB databases](#)” [*MobiLink - Getting Started*]

Finding CustDB sample files

The SQL Anywhere installer automatically installs CustDB when it installs the software. The following table lists the location of these files, as well as describes them.

Description	SQL Anywhere installation
<p><i>SQL Anywhere CustDB database:</i> The consolidated database. During installation, an ODBC data source named SQL Anywhere 10 CustDB is created for this database.</p> <p>☞ For information on the schema of this file, see “Exploring the CustDB Sample for MobiLink” [MobiLink - Getting Started].</p>	<p>The CustDB installation depends on whether you want to use the existing sample or recreate a new file:</p> <ul style="list-style-type: none"> ◆ For the existing sample: <code>samples-dir\UltraLite\CustDB\custdb.db</code> ◆ Erases changes that were synchronized into the consolidated <code>CustDB.db</code> file, so you have a clean version to work with: <code>samples-dir\UltraLite\CustDB\newdb.bat</code>
<p><i>The UltraLite CustDB database:</i> The remote version of the consolidated database that contains only a subset of the information, depending on which user synchronizes the database.</p>	<p>The file name and location can vary depending on the platform, programming language, or even device.</p> <ul style="list-style-type: none"> ◆ For AppForge: <code>samples-dir\UltraLiteforAppForge\</code>. You can find a CrossFire or MobileVB specific file version for the device you require. In most cases, the file name is <code>ul_CustDB.udb</code>. However, if you move the Palm version of this file to the desktop, it becomes <code>ul_CustDB.udb.pdb</code>. ◆ For UltraLite.NET: <code>samples-dir\UltraLite.NET\CustDB\Common\</code> ◆ For all other platforms and APIs: <code>samples-dir\UltraLite\CustDB\custdb.udb</code>
<p><i>RDBMS-specific build scripts:</i> The SQL scripts that rebuild a CustDB consolidated database for any one of the supported RDBMSs.</p> <p>☞ For more details on how to set up a consolidated database, see “Setting up the CustDB consolidated database” [MobiLink - Getting Started].</p>	<p>In the <code>samples-dir\MobiLink\CustDB</code> directory, you can find the following files:</p> <ul style="list-style-type: none"> ◆ For SQL Anywhere: <code>custdb.sql</code> ◆ For Adaptive Server Enterprise: <code>custase.sql</code> ◆ For Microsoft SQL Server: <code>custmss.sql</code> ◆ For Oracle: <code>custora.sql</code> ◆ For IBM DB2: <code>custdb2.sql</code>

Description	SQL Anywhere installation
<p><i>UltraLite CustDB client applications:</i> The end-user tools that provide a user-friendly interface to the UltraLite remote database. There is a sample client installed for each supported API.</p> <p>Each client application also contains a <i>readme.html</i> file that outlines important information you need to know about using that client sample.</p>	<p>The location varies depending on your development environment. Choose one of:</p> <ul style="list-style-type: none"> ◆ For Windows desktop: <i>install-dir\ultralite\CustDB\win32\386\</i>. ◆ For C/C++: Generic files are located in the <i>samples-dir\UltraLite\Custdb\</i> directory. Files specific to CodeWarrior for the Palm Computing Platform are in the following locations: <i>samples-dir\UltraLite\CustDB\cwcommon\</i> and <i>samples-dir\UltraLite\CustDB\cw\</i> directories. ◆ For embedded SQL: <i>samples-dir\UltraLite\CustDB\EVC\</i> and <i>samples-dir\UltraLite\CustDB\EVC40\</i> depending on your version of embedded Visual C++. ◆ For MobileVB: <i>samples-dir\UltraLiteForAppForge\CF_CustDB</i> or <i>samples-dir\UltraLiteForAppForge\MVB_CustDB</i> ◆ For .NET: <i>samples-dir\UltraLite.NET\CustDB</i> ◆ For M-Business Anywhere: <i>samples-dir\UltraLite-ForMBusinessAnywhere\CustDB</i>
<p><i>SQL synchronization logic:</i> The SQL statements needed to query and modify information from the UltraLite database and the calls required to start synchronization with the consolidated database.</p>	<p><i>samples-dir\UltraLite\CustDB\custdb.sqc</i></p>

Lesson 1: Logging in and populating the UltraLite remote

The following procedure starts the sample UltraLite client application, the sample MobiLink server, and populates the UltraLite remote database by synchronizing the UltraLite CustDB sample remote database to obtain an initial set of data from SQL Anywhere consolidated database. In this walkthrough the sample application is running on the same desktop computer as the MobiLink server. However, you can also deploy a client application to the device and achieve the same result.

The data you download depends on the user ID you enter when you start the application. By default, user ID 50 is used to log into UltraLite.

◆ To start and synchronize the sample application

1. Launch the MobiLink server sample.

At a command prompt enter:

```
mksrv10 -c "DSN=SQL Anywhere 10 CustDB" -zu+ -vcrs
```

Or, from the Start menu, choose Programs ► SQL Anywhere 10 ► MobiLink ► MobiLink Server Sample.

If the server starts, a console window appears displaying messages on the MobiLink server's status.

2. Launch the UltraLite client sample application.

From the Start menu, choose Programs ► SQL Anywhere 10 ► UltraLite ► Windows Sample Application.

3. Enter an employee ID.

Type **50** then press Enter.

After you enter the employee ID, the application synchronizes. The MobiLink server console window displays messages showing the synchronization taking place.

The default 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 application contains data.

A company name and a sample order should appear in the application window.

Lesson 2: Using the CustDB client application

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 only displays a subset of columns according to the user that has authenticated.

Columns in the table are displayed 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 is used to identify whether or not the row needs to be synchronized.

See also

- ◆ “Tables in the CustDB databases” [[MobiLink - Getting Started](#)]

Browsing orders

Browsing orders is accomplished in a similar method for each version of the UltraLite client 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.

◆ To browse orders

1. To scroll down the list of customers, click Next.
2. To scroll up through the list of customers, click Previous.

Adding an order

Adding an order is carried out in a similar way in each version of the UltraLite client application.

By adding an order, you have modified the data in your local UltraLite database. This data is not shared with the consolidated database until you synchronize.

◆ To add an order

1. Create a new order.
Choose Order ► New.
The Add New Order dialog appears.
2. Choose a customer from the list downloaded from the consolidated database.
From the Customer dropdown list choose **Basements R Us**. This customer does not have any current orders.
3. Choose a product from the list downloaded from the consolidated database.
From the Product dropdown list choose **Screwmaster Drill**. The price of this item is automatically entered in the Price field.

4. Enter the quantity and discount.
Type **20** in the **Quantity** field and type **5** (percent) in the **Discount** field.
5. Press Enter to add this order to the remote database as a row in the ULOrder table.

Changing the status of an order

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 are changing the status of it as well as adding additional note for the sales person to review. However, the data in the consolidated database is unchanged until you synchronize.

◆ To approve, deny, and delete orders

1. Approve the order for **Apple Street Builders**.
 - a. Click Previous to locate this customer.
 - b. Click Approve to approve the order.
 - c. In the Approve Order dialog, choose **Good work!** from the Note dropdown list.
 - d. Press Enter.
The order appears with a status of Approved.
2. Deny the order for Art's Renovations.
 - a. Go to the next order in the list, which is from Art's Renovations.
 - b. Click Deny to deny this order.
 - c. In the Deny Order dialog, choose **Discount is too high** from the Note dropdown list.
 - d. Press Enter.

The order appears with a status of Denied.

3. Delete the order for Awnings R Us.
 - a. Go to the next order in the list, which is from Awnings R Us.
 - b. Delete this order by choosing Order ► Delete.

You are asked to confirm the deletion. Click Yes.

The order is marked as deleted. However, the current data remains in the UltraLite remote until you synchronize changes to the consolidated database.

Lesson 3: Synchronizing with the CustDB consolidated database

For synchronization to take place, the MobiLink server must be running. If you have shut down your MobiLink server, restart it as described in “[Lesson 1: Logging in and populating the UltraLite remote](#)” on page 83.

The synchronization process for the sample application removes approved orders from your database.

◆ To synchronize the UltraLite remote

1. Choose File ► Synchronize to synchronize your data.
2. Confirm that synchronization took place.
 - ◆ At the remote database, you can confirm all required transactions occurred by checking that the approved order for **Apple Street Builders** is now deleted. Do this by browsing the orders to confirm the absence of this entry.
 - ◆ At the consolidated database, you can also confirm all required actions occurred by checking data in the consolidated database.

Confirming the synchronization at the consolidated database

You use Interactive SQL or Sybase Central to connect to the consolidated database and confirm that your changes were synchronized.

◆ To confirm the synchronization (Interactive SQL)

1. Connect to the consolidated database from Interactive SQL.
 - a. Choose Start ► Programs ► SQL Anywhere 10 ► SQL Anywhere ► Interactive SQL.
The Interactive SQL Connect dialog appears.
 - b. Select ODBC Data Source Name and choose SQL Anywhere 10 CustDB from the dropdown list.
2. Confirm the status change of the approved and denied orders.

To confirm that the approval and denial have been synchronized, issue the following statement.

```
SELECT order_id, status
FROM ULOrder
WHERE status IS NOT NULL
```

The results show that order 5100 is approved, and 5101 is denied.

3. Confirm that the deleted order has been removed.

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.

```
SELECT *  
FROM ULOrder  
WHERE order_id = 5102
```

◆ **To confirm the synchronization (Sybase Central)**

1. Launch Sybase Central.
Choose Programs ► Sybase ► SQL Anywhere 10 ► Sybase Central.
2. Connect to the consolidated CustDB database.
 - a. Choose Connections ► Connect with SQL Anywhere 10.
The Connect dialog appears.
 - b. Select ODBC Data Source Name and choose SQL Anywhere 10 CustDB from the dropdown list.
3. Browse to the ULOrder table and confirm your modifications:
 - a. Double-click Tables to display all tables.
 - b. Double-click the ULOrder table.
 - c. Click the Data tab and check that:
 - ◆ Check that order 5100 is approved, and 5101 is denied.
 - ◆ Check that order 5102 has been deleted.

Lesson 4: Browsing MobiLink synchronization scripts

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.

☞ For information on how synchronization has been implemented in CustDB, see “[Synchronization design](#)” [*MobiLink - Getting Started*].

See also

- ◆ “[Writing Synchronization Scripts](#)” [*MobiLink - Server Administration*]
- ◆ “[Introducing UltraLite as a MobiLink client](#)” [*MobiLink - Client Administration*]

Browsing the synchronization scripts

In addition to the tables, users, and publications, you can also use Sybase Central to browse the synchronization scripts that are stored in the consolidated database. Sybase Central is the primary tool for adding these 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.

☞ For more information on synchronization logic used in CustDB, see “[Synchronization logic source code](#)” [*MobiLink - Getting Started*].

See also

- ◆ “[Connection scripts](#)” [*MobiLink - Server Administration*]
- ◆ “[Table scripts](#)” [*MobiLink - Server Administration*]

◆ To browse the synchronization scripts

1. Start Sybase Central.
Choose Programs ► SQL Anywhere 10 ► Sybase Central.
2. Connect to the consolidated CustDB database.
 - a. Choose Connections ► Connect with MobiLink 10.
The Connect dialog appears.
 - b. Select ODBC Data Source Name and choose SQL Anywhere 10 CustDB.
Click OK.

3. Open the Connection Scripts folder.

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 the synchronization events, you can control the actions that are carried out during synchronization.

4. Open the Synchronized Tables folder, and open the ULCustomer table folder.

The right pane lists a set of scripts that are specific to this table, and their corresponding events. These scripts control the way that data in the ULCustomer table is synchronized with the remote databases.

What's next?

In addition to the CustDB application, tutorials are provided for each of the supported interfaces. For more information, see the following sections:

- ◆ **UltraLite C++** “Tutorial: Build an Application Using the C++ API” [*UltraLite - C and C++ Programming*].
- ◆ **UltraLite embedded SQL** “Tutorial: Build an Application Using Embedded SQL” [*UltraLite - C and C++ Programming*].
- ◆ **UltraLite for AppForge** “Tutorial: A Sample Application for AppForge MobileVB” [*UltraLite - AppForge Programming*].
- ◆ **UltraLite.NET** “Tutorial: Build an UltraLite.NET Application” [*UltraLite - .NET Programming*].
- ◆ **UltraLite for M-Business Anywhere** “UltraLite for M-Business Anywhere Quick Start” [*UltraLite - M-Business Anywhere Programming*]

Part III. UltraLite Database Reference

This part provides a reference for UltraLite database properties, options, connection parameters, and utilities.

CHAPTER 6

UltraLite Database Settings Reference

Contents

case property	94
checksum_level property	95
date_format property	97
date_order property	100
fips property	102
global_id option	104
max_hash_size property	106
ml_remote_id option	108
nearest_century property	109
obfuscate property	110
page_size property	112
precision property	114
scale property	116
time_format property	118
timestamp_format property	120
timestamp_increment property	123
utf8_encoding property	125

About this chapter

Properties are used to describe settings you require for your UltraLite database—whether it's a creation-time database property or a post-creation database option. This chapter includes an alphabetical list of all settings you can use to configure your UltraLite database.

case property

Description

Sets the case sensitivity of string comparisons in the UltraLite database.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: - o case=value
From Sybase Central Use any wizard that creates a database.	On the New Database Collation and Character Set page, select the Use Case-sensitive String Comparisons option.
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

Ignore, Respect

Default

Ignore

Usage

You cannot change the case of an existing database. Instead, you must create a new database.

The case sensitivity of the 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. Passwords are always case sensitive, regardless of the case sensitivity of the database.

See also

- ◆ [“Case sensitivity considerations” on page 33](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)
- ◆ [“UltraLite Load XML to Database utility \(ulload\)” on page 174](#)
- ◆ UltraLite for C/C++ and embedded SQL: [“ULCreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C++ only: [“CreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“ULDatabaseManager class” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite.NET: [“ULDatabaseManager members” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method createDatabase” \[UltraLite - M-Business Anywhere Programming\]](#)

checksum_level property

Description

Sets the level of checksum validation in the database.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: - o checksum=value
From Sybase Central Use any wizard that creates a database.	On the New Database Storage Settings page, select the Checksum Level for Database Pages option .
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

You can set three levels of checksum validation:

Value	Definition
0	Do not add checksums to database pages.
1	Add checksums to important database pages (for example indexes and synchronization status pages), but not row pages.
2	Add checksums to all database pages.

Default

0

Usage

Checksums are used to detect offline corruption, 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 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 Solutions.

If you unload and reload an UltraLite database with checksums enabled, the checksum level is preserved and restored.

See also

- ◆ [“Verifying page integrity with checksums” on page 37](#)
- ◆ [“Database page size considerations” on page 36](#)

- ◆ “Supported exit codes” on page 189
- ◆ “UltraLite Create Database utility (ulcreate)” on page 165
- ◆ “UltraLite Initialize Database utility (ulinit)” on page 172
- ◆ “UltraLite Load XML to Database utility (ulload)” on page 174
- ◆ UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for C++ only: “CreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for AppForge: “ULDatabaseManager class” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “ULDatabaseManager members” [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

date_format property

Description

Sets the default string format in which dates are retrieved from the database.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: - o date_format=value
From Sybase Central Use any wizard that creates a database.	On the New Database Options page, set the Date Format option .
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed date part values

A string using any of the following symbols:

Symbol	Description
<i>yy</i>	Two digit year.
<i>yyyy</i>	Four digit year.
<i>mm</i>	Two digit month, or two digit minutes if following a colon (as in <i>hh:mm</i>).
<i>mmm[m...]</i>	Character short form for months—as many characters as there are "m"s. An uppercase M causes the output to be made uppercase.
<i>d</i>	Single digit day of week, (0 = Sunday, 6 = Saturday).
<i>dd</i>	Two digit day of month. A leading zero is not required.
<i>ddd[d...]</i>	Character short form for day of the week. An uppercase D causes the output to be made uppercase.
<i>hh</i>	Two digit hours. A leading zero is not required.
<i>nn</i>	Two digit minutes. A leading zero is not required.
<i>ss[.ss.]</i>	Seconds and parts of a second.
<i>aa</i>	Use 12 hour clock. Indicate times before noon with AM.

Symbol	Description
<i>pp</i>	Use 12 hour clock. Indicate times after noon with PM.
<i>jjj</i>	Day of the year, from 1 to 366.

Default

YYYY-MM-DD, which corresponds to ISO date format specifications.

Usage

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, and then the A.M. or P.M. indicator (which could be localized) is 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.

Example

The following table illustrates usage options `date_format` settings, together with the output from a `SELECT CURRENT DATE` statement, executed on Thursday May 21, 2001.

<code>date_format</code> syntax used	Result returned
<i>yyyy/mm/ddddd</i>	2001/05/21/thu
<i>jjj</i>	141
<i>mmm yyyy</i>	may 2001

date_format syntax used	Result returned
<i>mm-yyyy</i>	05-2001

See also

- ◆ “UltraLite Create Database utility (ulcreate)” on page 165
- ◆ “UltraLite Initialize Database utility (ulinit)” on page 172
- ◆ “UltraLite Load XML to Database utility (ulload)” on page 174
- ◆ “Date considerations” on page 34
- ◆ UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for C++ only: “CreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for AppForge: “ULDATABASEMANAGER class” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “ULDATABASEMANAGER members” [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

date_order property

Description

Controls the interpretation for the order of months, days, and year date parts.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: - o date_order=value
From Sybase Central Use any wizard that creates a database.	On the New Database Options page, set the Date Order option .
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

MDY, YMD, DMY

Default

YMD, which corresponds to ISO date format specifications.

Usage

You cannot change the date order of an existing database. Instead, you must create a new database.

Example

Different values determine what the date of 10/11/12 is translated as:

Syntax used	Translation
MDY	Oct 11 1912
YMD	Nov 12 1910
DMY	Nov 10 1912

See also

- ◆ [“Date considerations” on page 34](#)
- ◆ [“date_format property” on page 97](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)
- ◆ [“UltraLite Load XML to Database utility \(ulload\)” on page 174](#)

- ◆ UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for C++ only: “CreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for AppForge: “ULDatabaseManager class” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “ULDatabaseManager members” [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

fips property

Description

Controls AES FIPS compliant data encryption, by using a Certicom certified cryptographic algorithm.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: -o fips=<i>boolean</i> Remember to include the KEY connection parameter in ulcreate's connection string. This sets the encryption key used for FIPS encryption.
From Sybase Central Use any wizard that creates a database.	On the New Database Storage Settings page, choose to encrypt the database with strong encryption by selecting the AES FIPS algorithm option. Remember to also set and confirm the encryption key.
From a client application Use the create database method.	<ul style="list-style-type: none"> ◆ On Palm OS, use the ULEnableFipsStrongEncryption method. ◆ On all other platforms, set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

All boolean values are supported. For example, true/false, yes/no, 1/0, and so on.

Default

0 (databases are not encrypted)

Usage

You cannot change the encryption setting for an existing database. Instead, you must create a new database.

Once you configure the key used for FIPS encryption, users connecting to this database must supply the key each time they connect.

Deploying FIPS

To deploy a FIPS-enabled application, include all appropriate libraries for your platform (for example, *ulfips.dll*). On Windows CE, you also need to run *setup.exe* from the *install-dir\ce\arm\fips* directory.

See also

- ◆ “Strong encryption” [[SQL Anywhere Server - Database Administration](#)]
- ◆ “Security considerations” on page 38
- ◆ “DBKEY connection parameter” on page 143
- ◆ “UltraLite Create Database utility (ulcreate)” on page 165
- ◆ “UltraLite Initialize Database utility (ulinit)” on page 172
- ◆ “UltraLite Load XML to Database utility (ulload)” on page 174

- ◆ [“Saving, retrieving, and clearing encryption keys on Palm OS”](#) [*UltraLite - C and C++ Programming*]
- ◆ [UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function”](#) [*UltraLite - C and C++ Programming*]
- ◆ [UltraLite for C++ only: “CreateDatabase function”](#) [*UltraLite - C and C++ Programming*]
- ◆ [UltraLite for AppForge: “ULDatabaseManager class”](#) [*UltraLite - AppForge Programming*]
- ◆ [UltraLite.NET: “ULDatabaseManager members”](#) [*UltraLite - .NET Programming*]
- ◆ [UltraLite for M-Business Anywhere: “Method createDatabase”](#) [*UltraLite - M-Business Anywhere Programming*]

global_id option

Description

Sets the database identification number.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use the ulinfo utility.	Use the following property syntax: -g <i>ID</i>
From Sybase Central Use the Database Options dialog by right-clicking the database name and choosing Options.	Select the global_database_ID option and type a new string in the Value field below the options table.
From a client application Use the set database ID method.	The method of setting this identification number varies according to the programming interface you are using.

Allowed values

The global database identifier in each deployed UltraLite application must be set to a unique, non-negative integer before default values can be assigned. These identification numbers uniquely identify the databases.

Default

The range of default values for a particular global autoincrement column is $pn + 1$ to $p(n + 1)$, where p is the partition size of the column and n is the global database identification number.

Usage

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. You do not need to create a new database.

Example

To autoincrement UltraLite database columns from 3001 to 4000, set the global ID to 3.

See also

- ◆ [“Global database ID considerations” on page 40](#)
- ◆ [“UltraLite Information utility \(ulinfo\)” on page 169](#)
- ◆ [“Using GLOBAL AUTOINCREMENT in UltraLite” \[MobiLink - Client Administration\]](#)
- ◆ [UltraLite for C/C++: “SetDatabaseID function” \[UltraLite - C and C++ Programming\]](#)
- ◆ [UltraLite for Embedded SQL: “ULSetDatabaseID function” \[UltraLite - C and C++ Programming\]](#)

- ◆ UltraLite for AppForge: “[Properties](#)” [[UltraLite - AppForge Programming](#)]
- ◆ UltraLite.NET: “[DatabaseID property](#)” [[UltraLite - .NET Programming](#)]
- ◆ UltraLite for M-Business Anywhere: “[Method setDatabaseID](#)” [[UltraLite - M-Business Anywhere Programming](#)]

max_hash_size property

Description

Sets the maximum default index hash size in bytes. UltraLite only uses as many bytes as required for the data type(s) of the column(s), up to this maximum.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: -o max_hash_size=value
From Sybase Central Use any wizard that creates a database.	On the New Database Storage Settings page, select the Maximum Hash Size for Indexes option .
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

0 to 32 bytes

Default

4 bytes

Usage

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, the does not hash row values.

You cannot change the hash size for an index after the index has been created. You can, however, override the default when you create a new index with the UltraLite Create Index wizard in Sybase Central, or with the WITH MAX SIZE clause in a CREATE INDEX or a CREATE TABLE statement.

If you declare your columns as DOUBLE, FLOAT, or REAL data types, no hashing is used. The hash size is always ignored.

See also

- ◆ [“Index performance considerations” on page 33](#)
- ◆ [“Indexes in UltraLite databases” on page 20](#)
- ◆ [“Working with UltraLite indexes” on page 65](#)
- ◆ [“Choosing an optimal hash size” on page 69](#)
- ◆ [“CREATE INDEX statement” on page 328](#)
- ◆ [“CREATE TABLE statement” on page 331](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)

- ◆ “UltraLite Initialize Database utility (ulinit)” on page 172
- ◆ “UltraLite Load XML to Database utility (uload)” on page 174
- ◆ UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for C++ only: “CreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for AppForge: “ULDatabaseManager class” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “ULDatabaseManager members” [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

ml_remote_id option

Description

A unique identifier in UltraLite that is used for MobiLink synchronization.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use the ulinfo utility.	Use the following property syntax: -r ID
From Sybase Central Use the Database Options dialog.	Select the ml_remote_ID option and type a new string in the Value field below the options table.
From a client application Use the set database option method.	The method of setting this identification value varies according to the programming interface you are using.

Allowed values

Any value that uniquely identifies the database for MobiLink synchronization.

Default

Null (no remote ID)

Usage

Each remote database must be represented by a single user ID in the consolidated database. This user ID must be granted REMOTE permissions to identify their user ID and address as a subscriber to publications.

If ml_remote_id is NULL, the MobiLink server automatically assigns a unique value on the first synchronization.

See also

- ◆ [“Remote ID considerations for MobiLink server synchronization” on page 40](#)
- ◆ [“Granting REMOTE permissions” \[SQL Remote\]](#)
- ◆ [“UltraLite Information utility \(ulinfo\)” on page 169](#)
- ◆ [UltraLite for C/C++: “SetDatabaseOption function” \[UltraLite - C and C++ Programming\]](#)
- ◆ [UltraLite for Embedded SQL: “ULSetDatabaseOptionString function” \[UltraLite - C and C++ Programming\]](#)
- ◆ [UltraLite for AppForge: “SetDatabaseOption method” \[UltraLite - AppForge Programming\]](#)
- ◆ [UltraLite.NET: “SetDatabaseOption method” \[UltraLite - .NET Programming\]](#)

nearest_century property

Description

Controls the interpretation of two-digit years in string-to-date conversions. It is used when converting from strings to dates or timestamps.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: - o nearest_century=value
From Sybase Central Use any wizard that creates a database.	On the New Database Options page, set the Nearest Century option .
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

Integer, between 0 and 100 inclusive

Default

50

Usage

You cannot change the nearest century of an existing database. Instead, you must create a new database.

The nearest_century setting is a numeric value that acts as a rollover point. Two digit years less than the value are converted to 20yy, while years greater than or equal to the value are converted to 19yy.

See also

- ◆ [“Nearest century conversion considerations”](#) on page 35
- ◆ [“UltraLite Create Database utility \(ulcreate\)”](#) on page 165
- ◆ [“UltraLite Initialize Database utility \(ulinit\)”](#) on page 172
- ◆ [“UltraLite Load XML to Database utility \(ulload\)”](#) on page 174
- ◆ UltraLite for C/C++ and Embedded SQL: [“ULCreateDatabase function”](#) [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for C++ only: [“CreateDatabase function”](#) [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for AppForge: [“ULDatabaseManager class”](#) [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: [“ULDatabaseManager members”](#) [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: [“Method createDatabase”](#) [*UltraLite - M-Business Anywhere Programming*]

obfuscate property

Description

Controls obfuscation of data in the database. Obfuscation is a form of simple encryption.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: -o obfuscate=boolean
From Sybase Central Use any wizard that creates a database.	On the New Database Storage Settings page, choose to encrypt the database by selecting the Use Simple Encryption (Obfuscation) option.
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Values

All boolean values are supported. For example, true/false, yes/no, 1/0, and so on.

Default

0 (databases are not obfuscated)

Usage

Simple encryption is equivalent to obfuscation and 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.

To render the database inaccessible without the correct encryption key requires that you set the database to strong encryption.

See also

- ◆ [“Simple encryption” \[SQL Anywhere Server - Database Administration\]](#)
- ◆ [“Security considerations” on page 38](#)
- ◆ [“DBKEY connection parameter” on page 143](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)
- ◆ [“UltraLite Load XML to Database utility \(ulload\)” on page 174](#)
- ◆ [UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ [UltraLite for C++ only: “CreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ [UltraLite for AppForge: “ULDatabaseManager class” \[UltraLite - AppForge Programming\]](#)
- ◆ [UltraLite.NET: “ULDatabaseManager members” \[UltraLite - .NET Programming\]](#)

- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

page_size property

Description

Defines the database page size.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: -o page_size=value
From Sybase Central Use any wizard that creates a database.	On the New Database Storage Settings page, select the appropriate byte value that corresponds to the allowed value list below.
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

1 K, 2 K, 4 K, 8 K, and 16 K

Default

4 K

Usage

You cannot change the page size of an existing database. Instead, you must create a new database.

Use k or K to denote kilobyte units. If you use any value other than the allowed values listed, the size is changed to the next larger size. If you do not specify a unit, bytes are assumed.

Example

To set the page size of the database to 8 KBs, you can configure this value as follows:

```
page_size=8k
```

or

```
page_size=8192
```

See also

- ◆ [“Database page size considerations” on page 36](#)
- ◆ [“CACHE_SIZE connection parameter” on page 128](#)
- ◆ [“RESERVE_SIZE connection parameter” on page 148](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)

- ◆ “UltraLite Load XML to Database utility (ulload)” on page 174
- ◆ UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for C++ only: “CreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for AppForge: “ULDatabaseManager class” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “ULDatabaseManager members” [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

precision property

Description

Specifies the maximum number of digits in decimal point arithmetic results.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: -o precision=<i>value</i>
From Sybase Central Use any wizard that creates a database.	On the New Database Options page, set the Precision option .
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

Integer, between 1 and 127, inclusive

Default

30

Usage

The precision is the total number of digits to the left and right of the decimal point. Use this property with the scale property to specify the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

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 DECIMAL(15,4). If scale is 2, the result is DECIMAL(15,2). In both cases, there is a possibility of overflow.

You cannot change the precision of an existing database. Instead, you must create a new database.

See also

- ◆ [“Decimal point position considerations” on page 36](#)
- ◆ [“scale property” on page 116](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)
- ◆ [“UltraLite Load XML to Database utility \(ulload\)” on page 174](#)

- ◆ UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for C++ only: “CreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for AppForge: “ULDatabaseManager class” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “ULDatabaseManager members” [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

scale property

Description

Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: - o scale=value
From Sybase Central Use any wizard that creates a database.	On the New Database Options page, set the Scale option.
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

Integer, between 0 and 127, inclusive, and less than the value specified for the precision database option

Default

6

Usage

You cannot change the scale of an existing database. Instead, you must create a new database.

The scale property specifies the minimum number of digits after the decimal point. Multiplication, division, addition, subtraction, and aggregate functions can all have results that exceed the maximum scale.

Use this property with the precision property, which is also used to determine the entire length of arithmetic results.

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.

See also

- ◆ [“Decimal point position considerations” on page 36](#)
- ◆ [“precision property” on page 114](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)

- ◆ “UltraLite Load XML to Database utility (ulload)” on page 174
- ◆ UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for C++ only: “CreateDatabase function” [*UltraLite - C and C++ Programming*]
- ◆ UltraLite for AppForge: “ULDatabaseManager class” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “ULDatabaseManager members” [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

time_format property

Description

Sets the format for times retrieved from the database.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: -o time_format=value
From Sybase Central Use any wizard that creates a database.	On the New Database Options page, set the Time Format option.
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

A string using any of the following symbols:

Symbol	Description
<i>hh</i>	Two digit hours (24 hour clock).
<i>nn</i>	Two digit minutes.
<i>mm</i>	Two digit minutes if following a colon (as in <i>hh:mm</i>).
<i>ss[s...]</i>	Two digit seconds plus optional fraction.

Default

HH:NN:SS.SSS

Usage

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.

Controlling zero-padding 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.

Example

If a transaction was executed at 3:30 PM and you used the default time_format syntax of *HH:NN:SS.SSS*, the result would be:

15:30:55.0

See also

- ◆ [“Time considerations” on page 34](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)
- ◆ [“UltraLite Load XML to Database utility \(ulload\)” on page 174](#)
- ◆ [UltraLite for C/C++ and Embedded SQL: “ULCreateDatabase function” \[*UltraLite - C and C++ Programming*\]](#)
- ◆ [UltraLite for C++ only: “CreateDatabase function” \[*UltraLite - C and C++ Programming*\]](#)
- ◆ [UltraLite for AppForge: “ULDatabaseManager class” \[*UltraLite - AppForge Programming*\]](#)
- ◆ [UltraLite.NET: “ULDatabaseManager members” \[*UltraLite - .NET Programming*\]](#)
- ◆ [UltraLite for M-Business Anywhere: “Method createDatabase” \[*UltraLite - M-Business Anywhere Programming*\]](#)

timestamp_format property

Description

Sets the format for timestamps that are retrieved from the database.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: -o timestamp_format=value
From Sybase Central Use any wizard that creates a database.	On the New Database Options page, set the Timestamp Format option .
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

A string using any of the following symbols:

Symbol	Description
<i>yy</i>	Two digit year.
<i>yyyy</i>	Four digit year.
<i>mm</i>	Two digit month, or two digit minutes if following a colon (as in <i>hh:mm</i>).
<i>mmm[m...]</i>	Character short form for months—as many characters as there are "m"s. An uppercase M causes the output to be made uppercase.
<i>d</i>	Single digit day of week, (0 = Sunday, 6 = Saturday).
<i>dd</i>	Two digit day of month. A leading zero is not required.
<i>ddd[d...]</i>	Character short form for day of the week. An uppercase D causes the output to be made uppercase.
<i>hh</i>	Two digit hours. A leading zero is not required.
<i>nn</i>	Two digit minutes. A leading zero is not required.
<i>ss[.ss.]</i>	Seconds and parts of a second.
<i>aa</i>	Use 12 hour clock. Indicate times before noon with AM.

Symbol	Description
<i>pp</i>	Use 12 hour clock. Indicate times after noon with PM.
<i>jjj</i>	Day of the year, from 1 to 366.

Default

YYYY-MM-DD HH:NN:SS.SSS

Usage

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, and then the A.M. or P.M. indicator (which could be localized) is 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 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.

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.

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
```

See also

- ◆ [“Timestamp considerations” on page 35](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)

- ◆ [“UltraLite Load XML to Database utility \(uload\)” on page 174](#)
- ◆ UltraLite for C/C++ and Embedded SQL: [“ULCreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C++ only: [“CreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“ULDatabaseManager class” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite.NET: [“ULDatabaseManager members” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method createDatabase” \[UltraLite - M-Business Anywhere Programming\]](#)

timestamp_increment property

Description

Limits the resolution of timestamp values. As timestamps are inserted into the database, UltraLite truncates them to match this increment. This value is useful when a DEFAULT TIMESTAMP column is being used as a primary key or row identifier.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: -o timestamp_increment=value
From Sybase Central Use any wizard that creates a database.	On the New Database Options page, set the Timestamp Increment option.
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

1 to 60,000,000,000 microseconds

Default

1 microsecond

Usage

You cannot change the timestamp increment of an existing database. Instead, you must create a new database.

Example

To store a value such as '2000/12/05 10:50:53:700', set this property to 100000. This value will truncate the timestamp after the first decimal place in the seconds component.

See also

- ◆ [“Timestamp considerations” on page 35](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)
- ◆ [“UltraLite Load XML to Database utility \(ulload\)” on page 174](#)
- ◆ UltraLite for C/C++ and Embedded SQL: [“ULCreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C++ only: [“CreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“ULDatabaseManager class” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite.NET: [“ULDatabaseManager members” \[UltraLite - .NET Programming\]](#)

- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

utf8_encoding property

Description

Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode.

Set by

UltraLite employs different methods for setting this property:

Context	Implemented by
From the command line Use any utility that creates a database.	Use the following property syntax: -o utf8_encoding=<i>boolean</i>
From Sybase Central Use any wizard that creates a database.	On the New Database Options page, set the UTF-8 option .
From a client application Use the create database method.	Set this property as one of the creation parameters for an API's create database method on the database manager class.

Allowed values

All boolean values are supported. For example, true/false, yes/no, 1/0, and so on.

Default

0, or databases are not UTF-8 encoded.

Usage

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 comprising two or more bytes are considered to be alphabetic. This means that you can use these characters in identifiers without requiring double quotes.

By encoding your database in UTF-8, UltraLite uses the UTF8BIN collation to sort characters. The UTF8BIN character set is not specific to any particular native language; no specific code page is associated with this character set. Consequently, you can synchronize data from multiple native languages to the same consolidated database. If you try synchronizing UTF-8 encoded characters into a consolidated table that does not support Unicode, a user error is reported.

See also

- ◆ [“Platform requirements for character set encoding” on page 32](#)
- ◆ [“Character considerations” on page 31](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Initialize Database utility \(ulinit\)” on page 172](#)
- ◆ [“UltraLite Load XML to Database utility \(ulload\)” on page 174](#)
- ◆ UltraLite for C/C++ and Embedded SQL: [“ULCreateDatabase function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C++ only: [“CreateDatabase function” \[UltraLite - C and C++ Programming\]](#)

- ◆ UltraLite for AppForge: “ULDatabaseManager class” [*UltraLite - AppForge Programming*]
- ◆ UltraLite.NET: “ULDatabaseManager members” [*UltraLite - .NET Programming*]
- ◆ UltraLite for M-Business Anywhere: “Method createDatabase” [*UltraLite - M-Business Anywhere Programming*]

CHAPTER 7

UltraLite Connection String Parameters Reference

Contents

CACHE_SIZE connection parameter	128
CON connection parameter	130
DBF connection parameter	131
CE_FILE connection parameter	133
NT_FILE connection parameter	135
PALM_FILE connection parameter	137
PALM_DB connection parameter	139
SYMBIAN_FILE connection parameter	140
DBN connection parameter	142
DBKEY connection parameter	143
PALM_ALLOW_BACKUP connection parameter	145
PWD connection parameter	146
RESERVE_SIZE connection parameter	148
START connection parameter	149
UID connection parameter	150

About this chapter

Connection parameters are keywords used in connection strings to open and describe connections with your UltraLite database. This chapter includes an alphabetical list of all of the supported connection keywords for UltraLite database connection strings.

CACHE_SIZE connection parameter

Description

Defines the size of the database cache.

Syntax

CACHE_SIZE=*number*{ **k** | **m** | **g** }

Default

Varies according to the platform you are connecting to:

- ◆ The Windows desktop default is 512 KB.
- ◆ The Windows CE default is 256 KB.
- ◆ The Symbian OS default is 128 KB.
- ◆ The Palm OS default is determined by the amount of memory available to the device, and the type of memory used (file-based or record-based).

Usage

If the cache size is not specified, or you set the size to 0, the default size is used. The default cache size is conservative. If your testing shows the need for better performance, you should increase the cache size.

You can specify the size in units of bytes. Use the suffix **k** or **K** to indicate units of kilobytes, the suffix **m** or **M** to indicate megabytes, and the suffix **g** or **G** to indicate gigabytes. If you do not specify a unit, bytes are assumed by default.

If you exceed the maximum cache size, it is automatically replaced with your platform's upper cache size limit. For example, on Symbian OS, the maximum cache size is 2 MB. Increasing the cache size beyond the size of the database, however, does not provide any performance improvement.

A large cache size can interfere with the number of other applications you can use.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes('), leading double quotes ("), or semi-colons(;).

Example

The following connection string fragment sets the cache size to 128 KB.

```
"CACHE_SIZE=128k"
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“page_size property” on page 112](#)
- ◆ [“RESERVE_SIZE connection parameter” on page 148](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)

- ◆ UltraLite for AppForge: “Connecting to an UltraLite database” [[UltraLite - AppForge Programming](#)]
- ◆ UltraLite for AppForge: “OpenConnection method” [[UltraLite - AppForge Programming](#)]
- ◆ UltraLite for M-Business Anywhere: “Connecting to an UltraLite database” [[UltraLite - M-Business Anywhere Programming](#)]
- ◆ UltraLite for M-Business Anywhere: “Method openConnection” [[UltraLite - M-Business Anywhere Programming](#)]
- ◆ UltraLite.NET: “Connecting to a database” [[UltraLite - .NET Programming](#)]
- ◆ UltraLite.NET: “Open method” [[UltraLite - .NET Programming](#)]

CON connection parameter

Applies to

Applications that require concurrent connections to the database.

Description

Names a connection, to make switching to it easier in multi-connection applications.

Syntax

CON=*name*

Default

No connection name.

Usage

The CON parameter is global to the application.

Do not use this 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 semi-colons(;).

Example

The following connection string fragment sets the first connection name to MyFirstCon.

```
"CON=MyFirstCon"
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“DBN connection parameter” on page 142](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

DBF connection parameter

Applies to

Typically used when targeting a single platform or using UltraLite administration tools.

Description

These parameters perform two different functions:

- ◆ When creating a database, this parameter names the new database file.
- ◆ When opening a connection, this parameter indicates which database file you want to load and connect to.

Alternate

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
- ◆ PALM_FILE
- ◆ NT_FILE
- ◆ SYMBIAN_FILE

Syntax

DBF=*ul-db*

Default

Depends on the platform:

- ◆ **On desktop platforms** If you do not specify an NT_FILE or DBF value, UltraLite sets the file name to *\UltraLiteDB\ulstore.udb*.
- ◆ **On Windows CE** If you do not specify a CE_FILE or DBF value, UltraLite sets the file name to *\UltraLiteDB\ulstore.udb*.
- ◆ **On Palm OS** If you do not specify a PALM_FILE or DBF value, UltraLite sets the file name to *ulstore.udb*.
- ◆ **On Symbian OS** If you do not specify a SYMBIAN_FILE or DBF value, UltraLite sets the file name to *ulstore.udb*.

Recommendation

Always explicitly specify the parameter; do not rely on default behavior.

Usage

If specified, the platform-specific parameters take precedence over DBF.

Because they are aliases, if DBF is used concurrently, the last one specified takes precedence.

The value of DBF must meet the file name requirements for the platform in question.

If you are deploying to a Windows CE device, UltraLite utilities and wizards can administer an UltraLite database on an attached CE device. To identify a file on a CE device, ensure specify the required absolute path and use the **wce:** prefix as illustrated by the Windows CE example below.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes('), leading double quotes ("), or semi-colons(;).

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 CE device, use the following connection string:

```
"DBF=wce:\UltraLite\MyULdb.udb"
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“Specifying file paths in a connection parameter” on page 50](#)
- ◆ [“Precedence of parameters for UltraLite administration tools” on page 49](#)
- ◆ [“DBN connection parameter” on page 142](#)
- ◆ [“CE_FILE connection parameter” on page 133](#)
- ◆ [“PALM_FILE connection parameter” on page 137](#)
- ◆ [“NT_FILE connection parameter” on page 135](#)
- ◆ [“SYMBIAN_FILE connection parameter” on page 140](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

CE_FILE connection parameter

Applies to

An UltraLite client application that connects to a Windows CE device in addition to another platform using the same connection string.

Alternate

If you are connecting from an UltraLite administration tool, or your connection object only connects to a Windows CE database, use DBF.

Description

This parameter performs two different functions:

- ◆ When creating a database, this parameter names the new database file.
- ◆ When opening a connection to an existing database, this parameter identifies the database.

Syntax

CE_FILE=*path\ce-db*

Default

The default depends on what you set:

- ◆ If you do not set a value for this connection parameter, then the value for the DBF parameter, if one has been set, is used.
- ◆ If you do not set a value for this parameter or the DBF parameter, then the default value is `\UltraLiteDB\ulstore.udb`.

Recommendation

Always explicitly specify the parameter; do not rely on the default behavior.

Usage

This parameter takes precedence over the DBF parameter.

The value of CE_FILE must meet the file name requirements for Windows CE.

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 semi-colons(;).

Example

The following example creates a new connection and identifies different database files for the desktop and Windows CE platforms:

```
Set Connection = DatabaseMgr.OpenConnection("DBF=d:\Dbfile.udb;CE_FILE=\myapp\nMyDB.udb")
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“Specifying file paths in a connection parameter” on page 50](#)
- ◆ [“Precedence of parameters for UltraLite administration tools” on page 49](#)
- ◆ [“DBF connection parameter” on page 131](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

NT_FILE connection parameter

Applies to

An UltraLite client application that connects to a desktop database in addition to another platform using the same connection string.

Alternate

If you are connecting from an UltraLite administration tool, or your connection object only connects to a desktop database, use DBF.

Description

This parameter performs two different functions:

- ◆ When creating a database, this parameter names the new database file.
- ◆ When opening a connection to an existing database, the parameter identifies the database.

Syntax

NT_FILE=*path\nt-db*

Default

The default depends on what you set:

- ◆ If you do not set a value for this connection parameter, then the value for the DBF parameter, if one has been set, is used.
- ◆ If you do not set a value for this parameter or DBF, then the default value is *\UltraLiteDB\ulstore.udb*.

Recommendation

Always explicitly specify the parameter; do not rely on default behavior.

Usage

This parameter takes precedence over the DBF 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 semi-colons(;).

Example

The following example creates a new connection and identifies different database files for the desktop, Palm OS, and Windows CE platforms:

```
Connection = DatabaseMgr.OpenConnection( "UID=JDoe;PWD=ULdb;  
NT_FILE=c:\test\MyTestDB.udb;CE_FILE=\database  
\MyCEDB.udb;PALM_FILE=MyPalmDB_MyCreatorID" )
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“Specifying file paths in a connection parameter” on page 50](#)
- ◆ [“Precedence of parameters for UltraLite administration tools” on page 49](#)
- ◆ [“DBF connection parameter” on page 131](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

PALM_FILE connection parameter

Applies to

An UltraLite client application that connects to a Palm device in addition to another platform using the same connection string.

Alternate

If you are connecting from an UltraLite administration tool, or your connection object only connects to a Palm OS database, use DBF.

Description

This parameter performs two different functions:

- ◆ When creating a database, this parameter names the new database file.
- ◆ When opening a connection to an existing database, this parameter identifies the database.

Syntax 1: record based stores

PALM_FILE=*name*

Syntax 2: file-based stores

PALM_FILE=*vfs*: [*volume-label*: | *volume-ordinal*:] *filename*

Default

The default depends on what you set:

- ◆ If you do not set a value for this connection parameter, then the value for the DBF parameter, if one has been set, is used.
- ◆ If you do not set a value for this parameter or the DBF parameter, then the default value is *ulstore.udb*.

Recommendation

Always explicitly specify the parameter; do not rely on default behavior.

Usage

Do not use this parameter with desktop administration utilities; use the DBF parameter instead while working with Palm databases.

The value of PALM_FILE must meet the file name requirements for Palm OS platforms.

If you are developing on AppForge, you also need to use the PALM_DB parameter.

This parameter takes precedence over the DBF parameter.

On file-based stores, always specify the absolute file path. If directories are specified in the full path name, they are created if they do not already exist.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes('), leading double quotes ("), or semi-colons(;).

Example

The following example creates a new connection and identifies different database files for the desktop, Palm OS, and Windows CE platforms:

```
Connection = DatabaseMgr.OpenConnection( "UID=JDoe;PWD=ULdb;  
NT_FILE=c:\test\MyTestDB.udb;CE_FILE=\database  
\MyCEDB.udb;PALM_FILE=MyPalmDB_MyCreatorID" )
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“Specifying file paths in a connection parameter” on page 50](#)
- ◆ [“Precedence of parameters for UltraLite administration tools” on page 49](#)
- ◆ [“Registering the Palm creator ID” \[*UltraLite - C and C++ Programming*\]](#)
- ◆ [“PALM_DB connection parameter” on page 139](#)
- ◆ [“DBF connection parameter” on page 131](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[*UltraLite - C and C++ Programming*\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[*UltraLite - C and C++ Programming*\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[*UltraLite - C and C++ Programming*\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[*UltraLite - AppForge Programming*\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[*UltraLite - AppForge Programming*\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[*UltraLite - M-Business Anywhere Programming*\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[*UltraLite - M-Business Anywhere Programming*\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[*UltraLite - .NET Programming*\]](#)
- ◆ UltraLite.NET: [“Open method” \[*UltraLite - .NET Programming*\]](#)

PALM_DB connection parameter

Applies to

Used with the PALM_FILE connection parameter when developing an UltraLite client application for Palm OS in AppForge.

Description

Set the correct UltraLite creator ID so that creator ID is used, rather than the AppForge creator ID.

Syntax

PALM_DB=*creator-ID*

Default

The AppForge creator ID.

Usage

The creator ID is a four character string. If you want to use a different creator ID than the AppForge creator ID, you must change the default value.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes('), leading double quotes ("), or semi-colons(;).

Example

The following example creates a new connection and identifies different database files for the desktop, Palm OS, and Windows CE platforms:

```
Set Connection = DatabaseMgr.OpenConnection("file_name=d:\MyDB.udb;  
PALM_FILE=MyPOSdb;PALM_DB=Syb3;CE_FILE=\myapp\MyCEdb.udb")
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“PALM_FILE connection parameter” on page 137](#)
- ◆ [PalmSource creator ID page](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

SYMBIAN_FILE connection parameter

Applies to

An UltraLite client application that connects to a Symbian device in addition to another platform using the same connection string.

Alternate

If you are connecting from an UltraLite administration tool, or your connection object only connects to a Symbian OS database, use DBF.

Description

This parameter performs two different functions:

- ◆ When creating a database, this parameter names the new database file.
- ◆ When opening a connection to an existing database, the parameter identifies the database.

Syntax

SYMBIAN_FILE=*symbian-db*

Default

The default depends on what you set:

- ◆ If you do not set a value for this connection parameter, then the value for the DBF parameter, if one has been set, is used.
- ◆ If you do not set a value for this parameter nor the DBF parameter, then the default value is *ulstore.udb*.

Recommendation

Always explicitly specify the parameter; do not rely on default behavior.

Usage

This parameter takes precedence over the DBF parameter.

The value of SYMBIAN_FILE must meet the file name requirements for Symbian OS platforms.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes('), leading double quotes ("), or semi-colons(;).

Example

The following example creates a new connection and identifies different database files for the desktop, Symbian OS, and Windows CE platforms:

```
Connection = DatabaseMgr.OpenConnection("UID=JDoe;PWD=ULdb;  
NT_FILE=c:\test  
\MyTestDB.udb;CE_FILE=MyCEdb.udb;SYMBIAN_FILE=MySymbianDB.udb")
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“Specifying file paths in a connection parameter” on page 50](#)
- ◆ [“Precedence of parameters for UltraLite administration tools” on page 49](#)
- ◆ [“DBF connection parameter” on page 131](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

DBN connection parameter

Applies to

Connections to a database that has already been opened.

Description

For applications that connect to more than one database, use this parameter to distinguish the databases by name.

Syntax

DBN=*db-name*

Default

Depends on the platform in question:

- ◆ **On Palm OS** The default value is the creator ID.
- ◆ **On all other platforms** The default value is derived from the file name value by removing the path and extension, if they exist. Cannot exceed 16 characters in length.

Usage

UltraLite only 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 semi-colons(;).

Example

Use the following parameters to successfully connect to the running UltraLite database named Kitchener:

```
DBN=Kitchener;DBF=cities.udb
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“DBF connection parameter” on page 131](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

DBKEY connection parameter

Applies to

For applications connecting to encrypted databases.

Description

Performs two different functions:

- ◆ When creating a database, this parameter provides an encryption key for the database.
- ◆ When opening a connection to an existing database, this parameter provides the same encryption key for the database. If the key provided is incorrect, the connection fails.

Syntax

DBKEY=*string*

Default

No key is provided.

Usage

If a database is created using an encryption key, the database file is strongly encrypted using either the FIPS or the AES 128-bit algorithm. The latter is the same algorithm used to encrypt SQL Anywhere databases. 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.

On Palm OS, applications are automatically shut down by the system whenever a user switches to a different application. However, you can program your UltraLite client to circumvent the need to re-enter the key each time a user switches back to the application again.

Any leading or trailing spaces in parameter values are ignored. This connection parameter's value cannot include leading single quotes('), leading double quotes ("), or semi-colons(;).

See also

- ◆ UltraLite for C/C++: “Saving, retrieving, and clearing encryption keys on Palm OS” [[UltraLite - C and C++ Programming](#)]
- ◆ “Opening connections with connection strings” on page 47
- ◆ “Security considerations” on page 38
- ◆ “obfuscate property” on page 110
- ◆ UltraLite for C/C++: “Connecting to a database” [[UltraLite - C and C++ Programming](#)]
- ◆ UltraLite for C/C++: “OpenConnection function” [[UltraLite - C and C++ Programming](#)]
- ◆ UltraLite for embedded SQL: “Connecting to a database” [[UltraLite - C and C++ Programming](#)]
- ◆ UltraLite for AppForge: “Connecting to an UltraLite database” [[UltraLite - AppForge Programming](#)]
- ◆ UltraLite for AppForge: “OpenConnection method” [[UltraLite - AppForge Programming](#)]
- ◆ UltraLite for M-Business Anywhere: “Connecting to an UltraLite database” [[UltraLite - M-Business Anywhere Programming](#)]

- ◆ UltraLite for M-Business Anywhere: “Method openConnection” [*UltraLite - M-Business Anywhere Programming*]
- ◆ UltraLite.NET: “Connecting to a database” [*UltraLite - .NET Programming*]
- ◆ UltraLite.NET: “Open method” [*UltraLite - .NET Programming*]

PALM_ALLOW_BACKUP connection parameter

Applies to

Desktop backup using the UltraLite HotSync conduit for Palm OS devices.

Description

Controls backup behavior over HotSync, which is disabled in UltraLite by default.

Syntax

PALM_ALLOW_BACKUP={ **yes** | **no** }

Usage

On Palm, you can back up the database to desktop using HotSync. In most UltraLite client applications, data is backed up by synchronization, so there is no need to use informal backups to the desktop. This is why the UltraLite runtime disables Palm's backup behavior. However, if your deployment explicitly requires that HotSync back up the UltraLite database to the desktop while it is also being synchronized, use this parameter to override the UltraLite default.

Once you have enabled backups for HotSync, you do not need to configure backups in ULDBUtil. Backups will occur with each synchronization attempt until you disable it.

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“UltraLite Data Management utility for Palm OS \(ULDBUtil\)” on page 185](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

PWD connection parameter

Description

Defines the password for a user ID that is used for authentication.

Syntax

PWD=*password_string*

Default

If you do not set *both* the UID and PWD, UltraLite opens connections with **UID=DBA** and **PWD=sql**.

Usage

You can set passwords to NULL or an empty string, but they cannot exceed the maximum length of 31 characters.

Every user of a database has a password. UltraLite supports up to four user ID/password combinations.

This connection parameter is not encrypted. However, UltraLite hashes the password before saving it. Therefore, you can only modify a password from Sybase Central.

Examples

The following connection string fragment supplies the user ID **DBA** and password **sql**.

```
"UID=DBA;PWD=sql"
```

The following connection string fragment supplies the user ID **DBA** and an empty password.

```
"UID=DBA;PWD= ' ' "
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“The role of user authentication” on page 45](#)
- ◆ [“Interpreting user ID and password combinations” on page 46](#)
- ◆ [“UID connection parameter” on page 150](#)
- ◆ UltraLite for C/C++: [“Authenticating users” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Authenticating users” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Authenticating users” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Authenticating users” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)

- ◆ UltraLite.NET: “Authenticating users” [*UltraLite - .NET Programming*]
- ◆ UltraLite.NET: “Connecting to a database” [*UltraLite - .NET Programming*]
- ◆ UltraLite.NET: “Open method” [*UltraLite - .NET Programming*]

RESERVE_SIZE connection parameter

Description

Allows you to pre-allocate the file system space required for your UltraLite database, without actually inserting any data.

Syntax

RESERVE_SIZE= *number*{ **k** | **m** | **g** }

Default

0, or no reserve size needed.

Usage

The value you supply can be any value from 0 to your maximum database size. Specify the size in units of bytes, by using the suffix k or K to indicate units of kilobytes, the suffix m or M to indicate megabytes, and the suffix g or G to indicate gigabytes. If you do not specify a unit, bytes are assumed by default.

If the RESERVE_SIZE value is smaller than the database size, UltraLite ignores the parameter.

Reserving file system space can improve performance slightly and also prevent out-of-storage memory failures because the database is never truncated. The database only grows when required as the application updates the database. The RESERVE_SIZE parameter reserves space by growing the database to the given reserve size on creation or startup. Therefore, you should consider both the metadata overhead and row packing required—in addition to the database raw data—before deriving the required reserve size. You should then run the database with test data and observe the database size.

Example

The following connection string fragment sets the reserve size to 128 KB so the system reserves that much for the database upon startup.

```
"RESERVE_SIZE=128K"
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“CACHE_SIZE connection parameter” on page 128](#)
- ◆ [“page_size property” on page 112](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

START connection parameter

Applies to

UltraLite engine client applications.

Description

Specifies the location of the UltraLite engine executable and then starts it.

Syntax

START=*path\uleng10.exe*

Default

No startline for the UltraLite engine is used.

Usage

Only supply a StartLine (START) connection parameter if you are connecting to an engine that is not currently running.

See also

- ◆ [“UltraLite Engine utility \(uleng10\)” on page 168](#)
- ◆ [“Choosing a data management component” on page 11](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Connecting to an UltraLite database” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Method openConnection” \[UltraLite - M-Business Anywhere Programming\]](#)
- ◆ UltraLite.NET: [“Connecting to a database” \[UltraLite - .NET Programming\]](#)
- ◆ UltraLite.NET: [“Open method” \[UltraLite - .NET Programming\]](#)

UID connection parameter

Description

The user ID with which you connect to the database. The value must be an authenticated user for the database.

Syntax

UID=*id_name*

Default

If you do not set *both* the UID and PWD, UltraLite opens connections with **UID=DBA** and **PWD=sql**.

Usage

You cannot set the UID to NULL or an empty string. Nor can the user ID exceed a maximum length of 31 characters.

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 in question and then add a new one.

The user ID is 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 semi-colons(;).

Example

The following connection string fragment supplies the user ID **DBA** and password **sql** for a database called *c:\MyOrders.udb*:

```
"UID=DBA;PWD=sql"
```

See also

- ◆ [“Opening connections with connection strings” on page 47](#)
- ◆ [“The role of user authentication” on page 45](#)
- ◆ [“Interpreting user ID and password combinations” on page 46](#)
- ◆ UltraLite for C/C++: [“Authenticating users” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for C/C++: [“OpenConnection function” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Authenticating users” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for embedded SQL: [“Connecting to a database” \[UltraLite - C and C++ Programming\]](#)
- ◆ UltraLite for AppForge: [“Authenticating users” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“Connecting to an UltraLite database” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for AppForge: [“OpenConnection method” \[UltraLite - AppForge Programming\]](#)
- ◆ UltraLite for M-Business Anywhere: [“Authenticating users” \[UltraLite - M-Business Anywhere Programming\]](#)

- ◆ UltraLite for M-Business Anywhere: “Connecting to an UltraLite database” [*UltraLite - M-Business Anywhere Programming*]
- ◆ UltraLite for M-Business Anywhere: “Method openConnection” [*UltraLite - M-Business Anywhere Programming*]
- ◆ UltraLite.NET: “Authenticating users” [*UltraLite - .NET Programming*]
- ◆ UltraLite.NET: “Connecting to a database” [*UltraLite - .NET Programming*]
- ◆ UltraLite.NET: “Open method” [*UltraLite - .NET Programming*]

CHAPTER 8

UltraLite Utilities Reference

Contents

Introduction to UltraLite utilities	154
Interactive SQL utility (dbisql)	155
SQL Preprocessor for UltraLite utility (sqlpp)	158
UltraLite AppForge Registry utility (ulafreg)	161
UltraLite HotSync Conduit Installation utility for Palm OS (ulcond10)	163
UltraLite Create Database utility (ulcreate)	165
UltraLite Engine utility (uleng10)	168
UltraLite Information utility (ulinfo)	169
UltraLite Initialize Database utility (ulinit)	172
UltraLite Load XML to Database utility (ulload)	174
UltraLite Engine Stop utility (ulstop)	177
UltraLite Synchronization utility (ulsync)	178
UltraLite Unload Database to XML utility (ulunload)	180
UltraLite Unload Old Database utility (ulunloadold)	183
UltraLite Data Management utility for Palm OS (ULDBUtil)	185
Supported extended options	186
Supported exit codes	189

About this chapter

This chapter provides reference information about UltraLite utility programs.

Introduction to UltraLite utilities

UltraLite includes a set of utilities that are designed to perform basic database administration activities from a command prompt. Many of these utilities share a similar functionality to those used by SQL Anywhere 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 chapter are case sensitive, unless otherwise noted. Type options *exactly* as they are displayed.

Interactive SQL utility (dbisql)

Applies to

Both SQL Anywhere and UltraLite databases. If you connected to an UltraLite database however, certain menu items that are SQL Anywhere-specific are not displayed in the interface. For example, Tools ► Lookup Procedure Name, or Tools ► Index Consultant.

Description

Executes SQL commands or runs command files provided. The syntax described below is specific to UltraLite usage.

For syntax specific to SQL Anywhere usage, see [“The Interactive SQL utility” \[SQL Anywhere Server - Database Administration\]](#).

Syntax

dbisql -c "*connection-string*" [*options*] [*dbisql-command* | *command-file*]

Option	Description
@data	Read options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used. If you want 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.
-c " <i>connection-string</i> "	Required. Connect to the database as identified in the DBF or file_name parameter of your <i>connection-string</i> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-codepage <i>number</i>	Specify a codepage to use when reading or writing files. The default code page is the default code page for the platform you are running on.
-d <i>delimiter</i>	Specify 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. Command delimiters are used for all connections in that Interactive SQL session, regardless of the setting stored in the database.
-d1	Echo 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.
-f <i>filename</i>	Open (but do not run) the file called <i>filename</i> . Enclose the file name 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 to the console 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.

Option	Description
-nogui	Run in command-prompt mode. If you specify either <i>dbisql-command</i> or <i>command-file</i> , then -nogui is assumed.
-onerror behavior	Control what happens if an error is encountered while reading data from the specified command file. Define one of the following supported <i>behavior</i> values: <ul style="list-style-type: none"> ◆ Stop Interactive SQL stops executing statements. ◆ Prompt Interactive SQL prompts the user to see if the user wants to continue. ◆ Continue The error is ignored and Interactive SQL continues executing statements. ◆ Exit Interactive SQL terminates. ◆ Notify_Continue The error is reporting, and the user is prompted to press Enter or click OK to continue. ◆ Notify_Stop The error is reported, and the user is prompted to press Enter or click OK to stop executing statements. ◆ Notify_Exit The error is reported and the user is prompted to press Enter or click OK to terminate Interactive SQL.
-q	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
-x	Scan commands but do not execute them. This is useful for checking long command files for syntax errors.
-ul	Connect to UltraLite databases by default. By default, Interactive SQL assumes that you are connecting to SQL Anywhere databases. When you specify the -ul option, the default becomes UltraLite connections. 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 dropdown list on the Connect dialog.
<i>dbisql-command command-file</i>	Execute the SQL command provided, or those specified in the <i>command-file</i> name. Alternatively, if you do not specify a <i>dbisql-command</i> or <i>command-file</i> , Interactive SQL enters interactive mode, where you can type a command into a command window.

Usage

Interactive SQL allows you to type SQL commands or run command files. It also provides feedback about the number of rows affected, the time required for each command, the execution plan of queries, and any error messages.

About code pages In UltraLite, collations include a code page plus a sort order. Therefore, code page numbers correspond to the number displayed as part of the UltraLite collation name. To see a list of supported collations (and its corresponding codepage), run `ulcreate -l` at a command prompt.

For example, on an English Windows XP computer, windowed programs use the 1252 (ANSI) code page. If you want Interactive SQL to read files created using the 297 (IBM France) code page, specify the following option: **-codepage 297**.

About exit codes 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).

In no GUI 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.

Example

The following command, entered at a command prompt, runs the command file `mycom.sql` against the `CustDB.udb` database for UltraLite, using the default user ID **DBA** and the password **sql**. If there is an error in the command file, the process terminates.

```
dbisql -ul -c DBF=CustDB.udb -onerror exit mycom.sql
```

See also

- ◆ “Using configuration files” [[SQL Anywhere Server - Database Administration](#)]
- ◆ “File Hiding utility (dbfhide)” [[SQL Anywhere Server - Database Administration](#)]
- ◆ “UltraLite Connection String Parameters Reference” on page 127
- ◆ “UltraLite Create Database utility (ulcreate)” on page 165
- ◆ “Supported exit codes” on page 189

SQL Preprocessor for UltraLite utility (sqlpp)

Applies to

Embedded SQL only for both SQL Anywhere and UltraLite.

Description

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 syntax described below is specific to UltraLite usage.

☞ For syntax specific to SQL Anywhere usage, see “SQL preprocessor” [[SQL Anywhere Server - Programming](#)].

Syntax

sqlpp [*options*] *esql-filename* [*output-filename*]

Option	Description
-d	Generate code that reduces data space size, but increases code size. Data structures are reused and initialized at execution time before use.
-e level	Flag non-conforming SQL syntax as a error. The parameters for <i>level</i> include: <ul style="list-style-type: none"> ◆ e flag syntax that is not entry-level SQL/92 syntax ◆ l flag syntax that is not intermediate-level SQL/92 syntax ◆ f flag syntax that is not full SQL/92 syntax ◆ t flag non-standard host variable types ◆ u flag features not supported by UltraLite ◆ w allow all supported syntax
-h	Set the maximum output line length before lines in the output .c file are split into multiple lines. 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 (that is, output lines are not split by default).
-k	Notify the preprocessor that the program to be compiled includes a user declaration of SQLCODE.
-n	Generate line number information in the C file by using #line directives in the appropriate places in the generated code. Use this option to the report source errors and to debug source on line numbers in the <i>esql-filename</i> file, rather than in the <i>output-filename</i> file.
-o	Not applicable to UltraLite.

Option	Description
-q	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
-r	Not applicable to UltraLite.
-s <i>string-length</i>	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.
-u	Required for UltraLite. Generate output specifically required for UltraLite databases.
-w <i>level</i>	Flag non-conforming SQL syntax as a warning. The parameters for <i>level</i> include: <ul style="list-style-type: none"> ◆ e flag syntax that is not entry-level SQL/92 syntax ◆ l flag syntax that is not intermediate-level SQL/92 syntax ◆ f flag syntax that is not full-SQL/92 syntax ◆ t flag non-standard host variable types ◆ u flag features not supported by UltraLite ◆ w allow all supported syntax
-x	Change multibyte strings to escape sequences, so that they can be passed through a compiler.
-z <i>collation-sequence</i>	Specify the collation sequence.

Usage

About output files 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 *sql*. 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*.

About collations 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 its corresponding codepage), run `ulcreate -l` at a command prompt.

Example

The following command preprocess the *srcfile.sql* embedded SQL file in quiet mode for an UltraLite application.

```
sqlpp -u -q MyEsqFile.sql
```

See also

- ◆ [“SQL Anywhere Embedded SQL” \[SQL Anywhere Server - Programming\]](#)
- ◆ [“Character considerations” on page 31](#)

UltraLite AppForge Registry utility (ulafreg)

Applies to

AppForge development environments only.

Description

Registers either the UltraLite runtime or the UltraLite Engine client, so that the development environment contains a reference to the appropriate UltraLite namespace. You can also unregister these *.DLL files with this utility.

Syntax

ulafreg [*options*]

Option	Description
-d	Display the location of the currently installed version of AppForge.
-q	Set the utility to run in quiet mode. Suppress informational banners, version numbers, and status messages. Error messages are still displayed, however.
-r [<i>directory</i>]	Register the UltraLite runtime *.DLL in named <i>directory</i> (for example, <i>ulmnb10.dll</i>). Only one application can reference this file. If you do not set <i>directory</i> , the default SQL Anywhere installation location is assumed (that is, <i>install-dir\ultralite\UltraLiteForAppForge\win32</i>).
-rc [<i>directory</i>]	Register the UltraLite Engine client component *.DLL in named <i>directory</i> (for example, <i>ulmnc10.dll</i>). Multiple applications can reference the same registered engine client *.DLL file. If you do not set <i>directory</i> , the default SQL Anywhere installation location is assumed (that is, <i>install-dir\ultralite\UltraLiteForAppForge\win32</i>).
-u	Unregister the UltraLite AppForge component. The 10.0 UltraLite for AppForge component replaces previous versions of UltraLite for MobileVB. You cannot use more than one version of the component on the same machine. Therefore you must unregister these previous version before registering an UltraLite for Appforge 10.0 component.

Usage

The behavior of this utility is forward compatible.

If you are unsure whether or not you need to run the utility, follow the procedure described below.

◆ To determine if you need to add UltraLite to your environment

1. Do one of the following:

- ◆ Check that a new project includes a reference to an UltraLite-specific namespace.

- ◆ Check if the correct connection class (for example, ULConnectionParms) appears in the AppForge list of available classes.
2. If it does not, close AppForge.
3. Run the ulafreg utility with the appropriate options.
4. Restart AppForge.

See also

- ◆ [“UltraLite Engine utility \(uleng10\)” on page 168](#)
- ◆ For MobileVB: [“Adding UltraLite to the MobileVB design environment” \[*UltraLite - AppForge Programming*\]](#)
- ◆ For Crossfire: [“Adding UltraLite to the Crossfire design environment” \[*UltraLite - AppForge Programming*\]](#)

UltraLite HotSync Conduit Installation utility for Palm OS (ulcond10)

Applies to

Palm OS databases that synchronize with HotSync Manager via the desktop.

Description

Installs and registers each database with the conduit so the conduit can manage HotSync synchronization operations for each database. You can also use this utility to uninstall the conduit.

Syntax

ulcond10 -c "connection-string" [options] creator-ID

Option	Description
<i>creator-ID</i>	Required. Set the creator ID of the application that uses the conduit. If a conduit already exists for the specified creator ID, it is replaced by the new conduit.
-a	Append additional database connection strings to the connection string configured with the -c option. Use this to register more than one database with the conduit.
-c "connection-string"	Required. Connect to the Palm database on the device as identified by the DBF parameter of your <i>connection-string</i> . The connection string you define registers the deployed Palm database with the conduit, and the connection parameters are stored as part of the conduit's configuration information. If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-d filename	Set the name of the plug-in <i>.dll</i> file. The utility creates a registry key (<i>Software\Sybase\SQL Anywhere\version\Conduit\creator-ID</i>) with a value named <i>PluginDLL</i> . The value for <i>PluginDLL</i> is the <i>filename</i> you set. Supply an empty string (-d"") to clear an existing file name from the registry.
-n name	Set the name displayed by the HotSync manager. The default value is Conduit . Do not use this option with the -u option.
-q	Set the utility to run in quiet mode. Suppress informational banners and version numbers.
-QQ	Set the utility to run in ultra-quiet mode. Suppress informational banners, version numbers, and status/error messages.

Option	Description
-u	Uninstall the conduit for the creator ID. If you do not specify -u , a conduit is installed.
-x sync-params	Set a list of semicolon-separated keyword=value pairs for the synchronization parameters you require. The keywords are case insensitive. See “Extended synchronization parameters” on page 187 . Supply an empty string (-x "") to clear any existing parameters.

Usage

HotSync Manager must be installed on your computer for the HotSync Conduit Installer to run.

HotSync records when each synchronization took place and whether each installed conduit worked as expected. The HotSync log file is in the subdirectory *User* of your Palm desktop installation directory.

Examples

The following command installs a conduit for the application with creator ID **Syb2**, named **CustDB**. These are the settings for the CustDB sample application:

```
ulcond10 -c "DBF=custdb.udb;UID=DBA;PWD=sql" -n CustDB Syb2
```

The following command uninstalls the conduit for the same CustDB sample application:

```
ulcond10 -u Syb2
```

See also

- ◆ [“UltraLite Connection String Parameters Reference” on page 127](#)
- ◆ [“UltraLite Data Management utility for Palm OS \(ULDBUtil\)” on page 185](#)
- ◆ [“Deploying the UltraLite HotSync conduit to the end-user's desktop” \[MobiLink - Client Administration\]](#)
- ◆ [“Network protocol options for UltraLite synchronization streams” \[MobiLink - Client Administration\]](#)
- ◆ [“Using HotSync on Palm OS” \[MobiLink - Client Administration\]](#)
- ◆ [“Registering the Palm creator ID” \[UltraLite - C and C++ Programming\]](#)

UltraLite Create Database utility (ulcreate)

Applies to

All UltraLite databases.

Description

Creates an UltraLite database with the properties you define.

Syntax

```
ulcreate -c "connection-string" [ options ] new-database-file
```

Option	Description
-c "connection-string"	Required. Create the database as identified in the DBF or file_name parameter of your <i>connection-string</i> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed. If you do not provide file name as a parameter in the connection string, ulcreate checks the end of the command for the file you specified as <i>new-database-file</i> .
-g <i>global-ID</i>	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.
-o [<i>extended-options</i>]	Set extended UltraLite database creation options. See “Extended creation-time options” on page 186 .
-l	List the available collation sequences and exit.
-ol	List the available extended database creation options and exit. See “Extended creation-time options” on page 186 .
-p <i>creator-ID</i>	Required for Palm OS. Create the database with the specified four character <i>creator-ID</i> of the UltraLite client application.
-q	Run in quiet mode—do not print messages.
-t <i>file</i>	Specify the file containing the public trusted root certificate. This certificate is required for server authentication.
-v	Print verbose messages.
-y	Overwrite the database file if it exists.
-z <i>collation-sequence</i>	Specify the label of the collation to be used.

Option	Description
<i>new-database-file</i>	Create a file with the specified name. Only use this standalone filename if you are not setting a connection string to set initial database parameters like a user ID (UID) or a password (PWD). Ensure the standalone filename you set is appropriate for your platform.

Usage

If you do not set any database properties, `ulcreate` creates a case insensitive, non-Unicode database with a collation sequence that depends on the current locale.

Case sensitivity Database passwords are always case sensitive, regardless of the case-sensitivity of the database. Database case sensitivity depends on whether you use the `case=respect` extended option setting.

About collations The collation sequence is used for all string comparisons in the database. In UltraLite, collations include a code page plus a sort order. If you do not specify `-z`, `ulcreate` attempts to determine a reasonable collation to use based on the current locale of the desktop.

To see a list of supported collations (and its corresponding codepage), run `ulcreate -l` at the command prompt.

UTF-8 encoding Determining whether or not to use the UTF-8 encoding depends on the operating system of your device.

Palm OS databases Palm databases written to the desktop must be identified with the `.pdb` extension. However, once you deploy the database to the device, the extension is dropped. For more details on file name formats, see [“Palm OS” on page 51](#).

Errors This utility returns error codes. Any value other than 0 means that the operation failed.

Example

Create an UltraLite database called `test.udb` as a case-insensitive, non-Unicode database with a collation sequence that depends on the current locale:

```
ulcreate test.udb
```

Create a case sensitive database called `test.udb` so that the database is created with ISO-compatible date formatting and ordering, by executing the following command:

```
ulcreate -c DBF=test.udb -o case=respect -o date_format=YYYY-MM-DD -o date_order=YMD
```

Create an encrypted database called `test.udb` with the `afvc_1835` encryption key:

```
ulcreate -c "DBF=test.udb;DBKEY=afvc_1835"
```

See also

- ◆ [“Creating and Configuring UltraLite Databases” on page 23](#)
- ◆ [“Specifying file paths in a connection parameter” on page 50](#)
- ◆ [“UltraLite Connection String Parameters Reference” on page 127](#)
- ◆ [“Supported exit codes” on page 189](#)
- ◆ [“Supported and alternate collations” \[SQL Anywhere Server - Database Administration\]](#)
- ◆ [“Registering the Palm creator ID” \[UltraLite - C and C++ Programming\]](#)

- ◆ “case property” on page 94
- ◆ “Platform requirements for character set encoding” on page 32

UltraLite Engine utility (uleng10)

Applies to

Windows XP and Windows CE.

Description

Manages UltraLite databases. Use the UltraLite engine to manage concurrent database connections from multiple UltraLite client applications.

To start the UltraLite engine from a client application, use the START connection parameter.

Usage

There are no options for the UltraLite engine.

The UltraLite engine does not display a console window on startup.

See also

- ◆ [“Choosing a data management component” on page 11](#)
- ◆ [“UltraLite Engine Stop utility \(ulstop\)” on page 177](#)
- ◆ [“START connection parameter” on page 149](#)

UltraLite Information utility (ulinfo)

Description

This utility performs the following functions:

- ◆ Displays information about an UltraLite database.
- ◆ Changes or clears the `global_id` or `ml_remote_id` database options.

Syntax

ulinfo -c "*connection-string*" [*options*]

Option	Description
-c " <i>connection-string</i> "	Required. Connect to the database as identified in the DBF or <code>file_name</code> parameter of your <i>connection-string</i> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-g <i>ID</i>	Set the initial global database ID to the value you assign. This value is used by the database for all new rows that have global autoincrement columns. The database uses this base value to autoincrement IDs associated with each additional row and/or column. When deploying an application, you must assign a different identification number to each database for synchronization with the MobiLink server.
-or	Open the database in read-only mode. UltraLite makes a copy of the original file, which you can then use to test your scripts without altering the database. Changes to the copied file are discarded upon completion. If you are connecting directly from the desktop to a database already deployed to a CE device, this option is not supported.
-q	Run in quiet mode—do not print messages.
-r <i>ID</i>	Set the initial <code>ml_remote_id</code> to the value you assign. By default, new UltraLite databases set the MobiLink remote ID to NULL. You can keep this default if you choose: both UltraLite and <code>dbmlsync</code> automatically set the MobiLink remote ID to a unique user ID (UUID) at the start of synchronization.
-rc	Clear the existing remote database ID and set the MobiLink remote ID to NULL.
-v	Print verbose messages. Display current database properties in addition to the database internals for the named database.

Usage

Warning messages generated when opening an UltraLite database are always displayed unless you use the `-q` option.

Example

Show basic database internals for a file named *sample.udb* that has already been synchronized:

```
ulinfo -c DBF=sample.udb

SQL Anywhere UltraLite Database Attribute Utility Version 10.0.0.XXXX Database
name: cv_dbattr Disk file 'run\script\cv_dbattr.udb'
Collation: 1252LATIN1
Number of Users: 1
  1. User: 'DBA'
Page size: 4096
Remote ID: JohnD
Global database ID: 1000
Global autoincrement usage: 0%
Number of tables: 3
Number of columns: 7
Number of publications: 3
Number of tables that will always be uploaded: 0
Number of tables that are never synchronized: 0
Number of primary Keys: 3
Number of foreign Keys: 0
Number of indexes: 0 Last synchronization completed successfully
Download occurred: 2005-12-07 15:01:28.615000
Upload OK
Upload rows not ignored
Partial download retained for subsequent sync
Actual MobiLink Authentication value: 1000
Authentication valid
Synchronization by publication number:
  1. Publication cv_sync
     Mask: 0x01
     Number of rows in next upload: 0
     Last download: 1900-01-01 00:00:00.000
  2. Publication cv_syncPub2
     Mask: 0x02
     Number of rows in next upload: 0
     Last download: 1900-01-01 00:00:00.000
  3. Publication cv_syncPub3
     Mask: 0x04
     Number of rows in next upload: 0
     Last download: 1900-01-01 00:00:00.000
```

Show database internals for a file named *CustDB.udb* and display database properties by enabling verbose messaging:

```
ulinfo -c DBF=CustDB.udb -v

SQL Anywhere UltraLite Database Attribute Utility Version 10.0.0.XXXX
Database information
Database name: custdb
Disk file 'C:\Documents and Settings\All Users\Shared Documents\SQL Anywhere
10\Samples\UltraLite\CustDB\custdb.udb'
Collation: 1252LATIN1
Number of Users: 1
  1. User: 'DBA'
Page size: 4096
Default index maximum hash size: 8
Checksum level: 0
MobiLink Remote ID: not set
Global database ID: not set
Encryption: None
Character encoding set: 1252LATIN1
```

```
Case sensitive: OFF
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: 6
Number of columns: 16
Number of publications: 1
Number of tables that will always be uploaded: 0
Number of tables that are never synchronized: 1
Number of primary Keys: 6
Number of foreign Keys: 2
Number of indexes: 3
This database has not yet been synchronized.
Synchronization by publication number:
  1. Publication test
      Mask: 0x01
      Number of rows in next upload: 0
      Last download: 1900-01-01 00:00:00.000
Done.
```

Clear the `ml_remote_id` for a file called `sample.udb`:

```
ulinfo -c DBF=sample.udb -rc
```

See also

- ◆ [“UltraLite Connection String Parameters Reference” on page 127](#)
- ◆ [“global_id option” on page 104](#)
- ◆ [“Global database ID considerations” on page 40](#)
- ◆ [“ml_remote_id option” on page 108](#)
- ◆ [“Remote ID considerations for MobiLink server synchronization” on page 40](#)


UltraLite Initialize Database utility (ulinit)

Description

Creates an UltraLite remote database from an existing SQL Anywhere consolidated database.

Syntax

```
ulinit -a "SAconnection-string" -c "ULconnection-string" -n pubname [ options ]
```

Option	Description
-a "SAconnection-string"	Required. Connect to the SQL Anywhere database specified in the <i>SAconnection-string</i> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-c "ULconnection-string"	Required. Connect to the database as identified in the DBF or file_name parameter of your <i>connection-string</i> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-n pubname	Required. Add tables to the UltraLite database schema. <i>pubname</i> specifies a publication in the reference database. Tables in the publication are added to the UltraLite database. The tables must already exist in the reference database; ulinit does not create them for you. 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* .
-o [extended-options]	Set the UltraLite extended database creation options. See “Extended creation-time options” on page 186 .
-p creator-ID	Required for Palm OS. Create the database with the specified four character <i>creator-ID</i> of the UltraLite client application.
-q	Run in quiet mode—do not print messages.
-s pubname	Use the named publication from the reference database to set synchronization behavior. Supply more than one -s option to name more than one synchronization publication. If -s is not supplied, the UltraLite remote has no named publications. To synchronize with all publications in the reference database, type -s* .  For more information on how to create publications for MobiLink synchronization, see “Publications in UltraLite” [MobiLink - Client Administration] .
-t file	Specify the file containing the trusted root certificate. This certificate is required for server authentication.
-w	Do not display warnings.

Usage

The SQL Anywhere reference database acts as the source for synchronization scripts, database configuration (for example, the collation sequence used), and table definitions for the UltraLite database file.

Alternate database creation methods If you want to create an UltraLite database without using a SQL Anywhere reference database, try one of the following methods:

- ◆ If you want to initialize an UltraLite database from an RDBMS other than SQL Anywhere, use the Create Synchronization Model wizard in Sybase Central. When you run the wizard, you are prompted to connect to a consolidated database to obtain schema information.
- ◆ If you want to create an empty UltraLite database that you can configure independent of any kind of reference database, use the `ulcreate` utility or the Create Database wizard for UltraLite.

UTF-8 encoding UltraLite typically uses the collation sequence defined in the reference database. However, you can still choose to use UTF-8 to encode the database, if your device requires encoded characters by setting the `utf8_encoding` property as part of your *extended-options* list.

However, there are cases where the `ulinit` utility restricts the collation sequences to those supported by UltraLite. To see a list of supported collations (and its corresponding codepage), run `ulcreate -l` at a command prompt. If your collation sequence is not supported by UltraLite, you should change it to one that is.

Palm OS databases Palm databases written to the desktop can be identified with the `.pdb` extension. However, once you deploy this file to the device, the extension is dropped. For more details on file name formats, see [“Palm OS” on page 51](#).

Examples

Create a file called `customer.udb` that contains the tables defined in TestPublication. For example:

```
ulinit -a "DSN=dbdsn;UID=JimmyB;PWD=secret" -c DBF=customer.udb -n
TestPublication
```

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. For example:

```
ulinit -a "DSN=dbdsn;UID=JimmyB;PWD=secret" -c DBF=customer.udb -n Pub1 -n
Pub2 -s Pub1 -s Pub2
```

Create a file called `customer.udb` for Palm OS using a registered creator ID. For example:

```
ulinit -a "DSN=dbdsn;UID=JimmyB;PWD=secret" -c DBF=customer.udb.pdb -n
TutCustomersPub
-p creator-id
```

See also

- ◆ [“Creating and configuring a model” \[MobiLink - Getting Started\]](#)
- ◆ [“Create Synchronization Model wizard” \[MobiLink - Getting Started\]](#)
- ◆ [“UltraLite Create Database utility \(ulcreate\)” on page 165](#)
- ◆ [“UltraLite Connection String Parameters Reference” on page 127](#)

UltraLite Load XML to Database utility (ulload)

Description

This utility performs the following functions:

- ◆ Creates a new database by loading data from an XML file.
- ◆ Loads data into an existing database.

Syntax

```
ulload -c "connection-string" [ options ] xml-file
```

Option	Description
-a	Append data and schema definitions into an existing database. If you are appending data into a pre-existing record-based database for Palm OS (that is, one with a <i>.pdb</i> extension), do not use the -p option.
-c "connection-string"	Required. Connect to the database as identified in the DBF or file_name parameter of your <i>connection-string</i> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-d	Load data only, ignoring any schema data in the XML file input.
-f directory	Set the XML file directory to an external path.
-g ID	Set the initial database ID to the INTEGER value you assign. This value is used by the database for all new rows that have global autoincrement columns. The database uses this base value to autoincrement IDs associated with each additional row and/or column. When deploying an application, you must assign a different identification number to each database for synchronization with the MobiLink server.
-i	Include inserted rows in the next upload synchronization. By default, rows inserted by this utility are not uploaded during synchronization.
-n	Load schema data only, ignoring any data in the XML file input.
-o [extended-options]	Set the UltraLite extended database creation options. See “Extended creation-time options” on page 186 .
-ol	List the available extended database creation options and exit. See “Extended creation-time options” on page 186 .

Option	Description
-onerror <i>behavior</i>	Control what happens if an error is encountered while reading data from the XML file. Define one of the following supported <i>behavior</i> values: <ul style="list-style-type: none"> ◆ continue The error is ignored and uload continues to load XML. ◆ prompt uload prompts you to see if you want to continue. ◆ quit uload stops loading the XML and terminates with an error. This is the default behavior if none is specified. ◆ exit uload exits.
-or	Open database in read-only mode. UltraLite makes a copy of the original file, and then use the copy to test your scripts without altering the database. Changes to the copied file are discarded upon completion. If you are connecting directly from the desktop to a database already deployed to a CE device, this option is not supported.
-p <i>creator-ID</i>	Required for Palm OS only when you are creating a new database from the loaded file. Create the database with the specified four character <i>creator-ID</i> of the UltraLite client application. If you are appending data into a pre-existing record-based database for Palm OS (that is, one with a <i>.pdb</i> extension), do not use this option with the <i>-a</i> option.
-q	Run in quiet mode—do not print messages.
-s <i>file</i>	Log the SQL statements used to load the database into the specified <i>file</i> .
-t <i>file</i>	Specify the file containing the trusted root certificate. This certificate is required for server authentication.
-v	Print verbose messages.
-y	Overwrite <i>xml-file</i> without confirmation.
<i>xml-file</i>	Specify the name of the XML file from which data is loaded.

Usage

The uload utility creates the UltraLite database file, if the named file does not already exist.

Setting an option or specifying a certificate on the command line overrides any settings in the *xml-file* that is processed by uload.

This utility returns error codes. Any value other than 0 means that the operation failed.

Palm OS databases Palm databases written to the desktop can be identified with the *.pdb* extension. However, once you deploy the file to the device, the extension is dropped. For more details on file name formats, see [“Palm OS” on page 51](#).

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
```

Load XML from a file named *test_data.xml* into the copy UltraLite makes of the database named *sample.udb*. Discard those changes upon completion. This allows you to check for errors in the XML data and correct them. When data loads successfully, you can run the command without the `-or` option to keep the XML updates.

```
ulload -or -c DBF=sample.udb -a test_data.xml
```

See also

- ◆ “Registering the Palm creator ID” [*UltraLite - C and C++ Programming*]
- ◆ “UltraLite Connection String Parameters Reference” on page 127
- ◆ “UltraLite Unload Database to XML utility (ulunload)” on page 180
- ◆ “Supported exit codes” on page 189
- ◆ “global_id option” on page 104
- ◆ “Global database ID considerations” on page 40

UltraLite Engine Stop utility (ulstop)

Applies to

Windows XP and Windows CE.

Description

Stops the UltraLite engine.

Syntax

ulstop

Usage

There are no options for this utility.

See also

- ◆ [“Choosing a data management component” on page 11](#)
- ◆ [“UltraLite Engine utility \(uleng10\)” on page 168](#)

UltraLite Synchronization utility (ulsync)

Description

Synchronizes an UltraLite database with a MobiLink server. It is a tool for testing synchronization during application development.

Syntax

ulsync -c "*connection-string*" [*options*]

Option	Description
-a <i>authenticate-parameters</i>	Set any values required by your MobiLink server <i>authenticate_parameters</i> script. These may be a user name and password, for example. You can repeat this option to a maximum of 20 times, if required.
-c " <i>connection-string</i> "	Required. Connect to the database as identified in the DBF or file_name parameter of your <i>connection-string</i> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-e <i>sync-parms</i>	Set a single extended synchronization option. You can set multiple extended synchronization options with multiple -e options. The keywords are case insensitive. Note that you must always set Username and Version. For example: -e Username=MyName -e Version=MyNum See “Extended synchronization parameters” on page 187 .
-k <i>stream-type</i>	Specify the synchronization stream, which can be encrypted or unencrypted. The <i>stream-type</i> must be one of <i>tcpip</i> , <i>tls</i> , <i>http</i> , or <i>https</i> . The default stream is <i>tcpip</i> . For Palm OS, you must store the RSA certificate in the UltraLite database when it is created. For all other systems, you can specify the certificate file either with the -x trusted certificate=path option, or as an unnamed file with a certificate lookup mechanism that is part of the encryption code.
-n	Do not carry out synchronization. Use this option with -r to see previous synchronization results.
-or	Synchronize the database in read-only mode. UltraLite makes a copy of the original file, and then use the copy to test your scripts without altering the database. Changes to the copied file are discarded upon completion. If you are connecting directly from the desktop to a database already deployed to a CE device, the parameter is not supported.
-r	Display last synchronization results.
-q	Run in quiet mode—do not print messages.
-v	Display synchronization progress messages.

Option	Description
-x <i>netopt-string</i>	A semicolon-separated list of synchronization network protocol options (also known as stream parameters) and their respective values. See “ Network protocol options for UltraLite synchronization streams ” [<i>MobiLink - Client Administration</i>]. The default value is host=localhost .

Usage

Do not confuse the *sync-parms* option (which sets synchronization parameters) with the **-x network protocol options** option (which sets synchronization *stream* parameters).

This utility returns error codes. Any value other than 0 means that the operation failed.

Example

The following command synchronizes a database file named *myuldb.udb*. The MobiLink user name is **remoteA**.

```
ulsync -c DBF=myuldb.udb -k http -e "Username=remoteA;Version=2"
```

The following command uploads changes for a database file named *myuldb.udb* over HTTPS with the *c:\certs\rsa.crt* certificate. The MobiLink user name is **remoteB**.

```
ulsync -c DBF=myuldb.udb -k https -x trusted_certificate=c:\certs\rsa.crt  
-e "Username=remoteB;Version=2;UploadOnly=ON"
```

The following command displays the last synchronization results for a database file named *synced.udb*.

```
ulsync -n -r -c dbf=synced.udb
```

The previous synchronization results are listed as follows:

```
SQL Anywhere UltraLite Database Synchronize Utility Version 10.0  
Results of last synchronization:  
Succeeded  
  Download timestamp: 2006-07-25 16:39:36.708000  
  Upload OK  
  No ignored rows  
  Partial download retained  
  Authentication value: 1000 (0x3e8)
```

See also

- ◆ “[UltraLite Connection String Parameters Reference](#)” on page 127
- ◆ “[UltraLite Clients](#)” [*MobiLink - Client Administration*]
- ◆ “[Supported exit codes](#)” on page 189
- ◆ “[MobiLink file transfer utility \[mlfiletransfer\]](#)” [*MobiLink - Client Administration*]

UltraLite Unload Database to XML utility (ulunload)

Applies to

All UltraLite databases.

Description

Unloads any of the following, depending on the options used:

- ◆ An entire UltraLite database to XML.
- ◆ All or part of UltraLite data only to XML.
- ◆ An UltraLite schema only to XML.
- ◆ An UltraLite schema only as SQL statements.

Syntax

ulunload -c "connection-string" [options] output-file

Option	Description
-b max-size	Set the maximum size of column data to be stored in the XML file. The default is 10 K. To store all data in the XML file (that is, to have no maximum size), use -b -1 .
-c "connection-string"	Required. Connect to the database as identified in the DBF or file_name parameter of your <i>connection-string</i> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-d	Unload data only, ignoring any schema information in the database. Do not use this option if you intend to reload the XML file into a new database using the ulload utility.
-e table,...	Exclude data in the named <i>table</i> . You can name multiple tables in a comma-separated list. For example: <code>-e mydbtable1,mydbtable5</code>
-f directory	Set the directory to store data larger than the maximum size specified by -b . The default is the same directory as the XML file.
-n	Unload schema only, ignoring any data in the database.
-or	Open the database in read-only mode. UltraLite makes a copy of the original file, which you can then use to test your scripts without altering the database. Changes to the copied file are discarded upon completion. If you are connecting directly from the desktop to a database already deployed to a CE device, the parameter is not supported.
-q	Run in quiet mode. Do not print messages.

Option	Description
-s	Unload schema only as SQL Anywhere-compatible SQL statements; ignore any data in the database.
-t <i>table</i> ,...	Unload data in the named <i>table</i> only. You can name multiple tables in a comma separated list. For example: <code>-t mydbtable2,mydbtable6</code>
-v	Print verbose messages.
-x <i>owner</i>	Output tables so they are owned by a specific user ID. You can use this option with the -s option.
-y	Overwrite <i>output-file</i> without confirmation.
<i>output-file</i>	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.

Usage

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.

Preservation 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.

Column data overflows 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, or in the directory specified by -f. The file follows this naming convention:

`tablename-columnname-rownumber.bin`

Assigning table ownership 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. Do not use this option if you are using the resulting SQL to create or modify an UltraLite database.

Errors This utility returns error codes. Any value other than 0 means that the operation failed.

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 the *sample.xml* file. Overwrite the database if it exists.

```
ulunload -c DBF=sample.udb -d -y sample.xml
```

See also

- ◆ [“UltraLite Connection String Parameters Reference” on page 127](#)
- ◆ [“Supported exit codes” on page 189](#)
- ◆ [“UltraLite Load XML to Database utility \(ulload\)” on page 174](#)
- ◆ [“UltraLite Information utility \(ulinfo\)” on page 169](#)

UltraLite Unload Old Database utility (ulunloadold)

Applies to

UltraLite versions 8.0.2 to 9.0.x.

Description

Unloads previous versions of UltraLite databases and/or schema files (*.usm) into an XML file.

Syntax

ulunloadold -c "*connection-string*" [*options*] *xml-file*

Option	Description
-b <i>max-size</i>	Set the maximum size of column data to be stored in the XML file. The default is 10 K. To store all data in the XML file (that is, to have no maximum size), use -b -1 .
-c " <i>connection-string</i> "	Required. Connect to the database as identified in the DBF or file_name parameter of your <i>connection-string</i> . If you do not specify both a user ID and a password, the default UID of DBA and PWD of sql are assumed.
-f <i>directory</i>	Set the directory to store data larger than the maximum size specified by -b. The default is the same directory as the XML file.
-q	Run in quiet mode. Do not print messages.
-v	Print verbose messages.
-y	Overwrite <i>xml-file</i> without confirmation.
<i>xml-file</i>	Set the name of the XML file that data will be unloaded into.

Usage

Do not unload UltraLite version 10 databases with this utility. Use the ulunload utility instead. To create an UltraLite 10 version of the database, load the *xml-file* with ulload.

Preservation 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.

Column data overflows 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, or in the directory specified by -f. The file follows this naming convention:

tablename-columnname-rownumber.bin

Errors This utility returns error codes. Any value other than 0 means that the operation failed.

Example

Upgrading an UltraLite 8.0.x schema file named *dbschema8.usm* into an UltraLite version 10 database named *db.udb* requires these two commands:

```
ulunloadold -c SCHEMA_FILE=dbschema8.usm dbschema.xml
ulload -c DBF=db.udb dbschema.xml
```

Upgrading an UltraLite version 9.0.x database for Palm OS named *palm9db.pdb* to an UltraLite version 10 database named *palm10db.pdb* requires these two commands:

```
ulunloadold -c DBF=palm9db.pdb dbdata.xml
ulload -c DBF=palm10db.pdb -p Syb dbdata.xml
```

See also

- ◆ [“UltraLite Connection String Parameters Reference” on page 127](#)
- ◆ [“UltraLite Load XML to Database utility \(ulload\)” on page 174](#)
- ◆ [“UltraLite Unload Database to XML utility \(ulunload\)” on page 180](#)
- ◆ [“UltraLite Information utility \(ulinfo\)” on page 169](#)

UltraLite Data Management utility for Palm OS (ULDBUtil)

Applies to

Palm OS databases managed from the device.

Description

Once you install ULDBUtil on the device, you can use this utility to perform the following functions:

- ◆ Delete the database from the device when the device is shared among different users. Deleting the file allows you to save space or to maintain privacy. You can then reinstall the database or even have the database create a new unpopulated database.
- ◆ Back up the database upon next synchronization. Use this feature to perform an initial synchronization, and then back up the database. This allows you to deploy the database to other devices, so they do not need to perform an initial synchronization.

Usage

This utility is installed as the following file:

install-dir\UltraLite\Palm\68k\ULDBUtil.prc

◆ To delete UltraLite application data from a Palm OS device

1. Switch to ULDBUtil.
2. If your device has expansion cards, pick the media (that is, internal/external volume or record-based) from which the application database file is to be deleted.
3. From the list of UltraLite version 10 applications and databases, select an application.
4. Tap Delete on the Palm device to remove the data.

◆ To back up the database from a Palm OS device to a desktop

1. Switch to ULDBUtil.
2. Select the Backup option so HotSync knows to back up the database on the next synchronization attempt. You need to select this option for each subsequent back up you need.

Tip

If you want to enable automatic backups and thereby avoid the need to select this option each time you synchronize, use the PALM_ALLOW_BACKUP parameter.

See also

- ◆ [“UltraLite HotSync Conduit Installation utility for Palm OS \(ulcond10\)” on page 163](#)
- ◆ [“PALM_ALLOW_BACKUP connection parameter” on page 145](#)

Supported extended options

You can specify parameters for the UltraLite command line utilities. There are two types of parameters:

- ◆ Database creation options.
- ◆ Synchronization options.

See also

- ◆ [“Extended creation-time options” on page 186](#)
- ◆ [“Extended synchronization parameters” on page 187](#)

Extended creation-time options

The extended options supported for database creation utilities like `ulcreate`, `ulinit`, and `ulload` are the same creation-time properties you can set when configuring any new database. You can define extended database creation-time options in one of two ways:

- ◆ **Prepend each extended option with `-o`** For example, if you want to create a database with `ulcreate` so that the database is created with case sensitivity enabled and uses ISO-compatible date formatting and ordering, run the command as follows:

```
ulcreate -o case=respect -o date_format=YYYY-MM-DD
-o date_order=YMD
```

- ◆ **Assemble options into a creation-string** An assembly of creation parameters is called a **creation-string**. You type a creation-string on a single line with the parameter names and values separated by semicolons:

```
parameter1=value1; parameter2=value2
```

The UltraLite runtime ensures the creation-time parameters are assembled into a creation-string before creating a database with it. For example, if you use the `ulcreate` utility, the following connection string is used to create a new database with case sensitivity enabled and that uses ISO-compatible date formatting and ordering:

```
ulcreate - o "case=respect;date_format=YYYY-MM-DD;
date_order=YMD"
```

You must always configure database creation properties at creation time. You cannot alter these properties once the file has been written.

Property list	Description
case	Sets the case-sensitivity of string comparisons in the UltraLite database. See “Case sensitivity considerations” on page 33 and “case property” on page 94 .
checksum_level	Sets the level of checksum validation in the database. See “Verifying page integrity with checksums” on page 37 and “checksum_level property” on page 95 .

Property list	Description
date_format	Sets the default string format in which dates are retrieved from the database. See “Date considerations” on page 34 and “date_format property” on page 97 .
date_order	Controls the interpretation of date ordering of months, days, and years. See “Date considerations” on page 34 and “date_order property” on page 100 .
fips	Controls AES FIPS compliant data encryption, by using a Certicom certified cryptographic algorithm. FIPs encoding is a form of strong encryption. See “Security considerations” on page 38 and “fips property” on page 102 .
max_hash_size	Sets the default index hash size in bytes. See “Index performance considerations” on page 33 and “max_hash_size property” on page 106 .
nearest_century	Controls the interpretation of two-digit years in string-to-date conversions. See “Nearest century conversion considerations” on page 35 and “nearest_century property” on page 109 .
obfuscate	Controls whether or not to obfuscate data in the database. Obfuscation is a form of simple encryption. See “Security considerations” on page 38 and “obfuscate property” on page 110 .
page_size	Defines the database page size. See “Database page size considerations” on page 36 and “page_size property” on page 112 .
precision	Specifies the maximum number of digits in decimal point arithmetic results. See “Decimal point position considerations” on page 36 and “precision property” on page 114 .
scale	Specifies the minimum number of digits after the decimal point when an arithmetic result is truncated to the maximum precision. See “Decimal point position considerations” on page 36 and “scale property” on page 116 .
time_format	Sets the format for times retrieved from the database. See “Time considerations” on page 34 and “time_format property” on page 118 .
timestamp_format	Sets the format for timestamps retrieved from the database. See “Timestamp considerations” on page 35 and “timestamp_format property” on page 120 .
timestamp_increment	Determines how the timestamp is truncated in UltraLite. See “Timestamp considerations” on page 35 and “timestamp_increment property” on page 123 .
utf8_encoding	Encodes data using the UTF-8 format, 8-bit multibyte encoding for Unicode. See “Character considerations” on page 31 and “utf8_encoding property” on page 125 .

☞ For more information, see [“Choosing creation-time database properties” on page 30](#).

Extended synchronization parameters

You specify extended synchronization parameters with the `ulsync` utility on the command line after you have defined all other command line options you want to use. You can define multiple parameters by using

multiple -e options. You cannot build a semi-colon delimited keyword string. The keywords are case insensitive.

Option name	Description
DownloadOnly (boolean)	Specifies that synchronization should be download-only. See “ Download Only synchronization parameter ” [<i>MobiLink - Client Administration</i>].
Newpassword (string)	Specifies a new password. See “ New Password synchronization parameter ” [<i>MobiLink - Client Administration</i>].
Password (string)	Specifies a password for the MobiLink client whose subscriptions are being synchronized. See “ Password synchronization parameter ” [<i>MobiLink - Client Administration</i>].
Ping (boolean)	Confirms communications between the UltraLite client and the MobiLink server. See “ Ping synchronization parameter ” [<i>MobiLink - Client Administration</i>].
Publication (string)	Specifies the names publications to be synchronized. See “ Publication synchronization parameter ” [<i>MobiLink - Client Administration</i>].
PublicationMask (integer)	Names a set of publications for which the last download time is retrieved. The set is supplied as a mask. See “ PublicationMask synchronization parameter ” [<i>MobiLink - Client Administration</i>].
SendColumnNames (boolean)	Used to directly handle rows. See “ Send Column Names synchronization parameter ” [<i>MobiLink - Client Administration</i>].
SendDownloadACK	Specifies that a download acknowledgment should be sent from the client to the server. See “ Send Download Acknowledgment synchronization parameter ” [<i>MobiLink - Client Administration</i>].
TableOrder (string)	Sets the table order required for priority synchronization, if the default table ordering (based on foreign keys) is not suitable for your deployment. See “ Table Order synchronization parameter ” [<i>MobiLink - Client Administration</i>].
UploadOnly (boolean)	Specifies that synchronization should be upload-only. See “ Upload Only synchronization parameter ” [<i>MobiLink - Client Administration</i>].
Username (string)	Required. Defines the consolidated database user. See “ User Name synchronization parameter ” [<i>MobiLink - Client Administration</i>].
Version (string)	Required. Defines the consolidated database version. See “ Version synchronization parameter ” [<i>MobiLink - Client Administration</i>].

Supported exit codes

The ulcreate, ulload, ulsync, and ulunload utilities return exit codes to indicate whether or not the operation a utility attempted to complete was successful or not. 0 indicates a successful synchronization. Any other value indicates that the synchronization failed.

Exit code	Status	Description
0	EXIT_OKAY	Operation successful.
1	EXIT_FAIL	Operation failure.
3	EXIT_FILE_ERROR	Database cannot be found.
4	EXIT_OUT_OF_MEMORY	Exhausted all of the device's dynamic memory.
6	EXIT_COMMUNICATIONS_FAIL	Communications error generated while talking to the UltraLite engine.
9	EXIT_UNABLE_TO_CONNECT	Invalid UID or PWD provided, therefore cannot connect to the database.
12	EXIT_BAD_ENCRYPT_KEY	Missing or invalid encryption key.
13	EXIT_DB_VER_NEWER	Detected that the database version is incompatible. The database must be upgraded to a newer version.
255	EXIT_USAGE	Invalid command line options.

CHAPTER 9

UltraLite System Table Reference

Contents

UltraLite system tables 192

About this chapter

This chapter lists the system tables used by UltraLite. This catalogue of tables act as the schema by describing the metadata that define the structure of your UltraLite database.

UltraLite system tables

The schema of every UltraLite database is described in a number of system tables. Because UltraLite does not support table ownership, system tables are available to any of the four UltraLite user IDs.

The contents of these tables can be changed only by UltraLite itself. Therefore, UPDATE, DELETE, and INSERT commands cannot be used to modify the contents of these tables. Further, the structure of these tables cannot be changed using the ALTER TABLE and DROP commands.

You can browse the contents of these tables via Sybase Central.

◆ To hide or show system objects (Sybase Central)

1. Connect to the database.
2. Browse the objects of the database.
3. Right-click the contents pane and choose Hide/Show System Objects.
4. Click the Tables folder and browse the available tables.

systable system table

Description

Each row in the systable system table describes one table in the database.

Columns

Column name	Description	Type	Data constraint	Integrity constraints
column_count	The number of columns in the table.	UN-SIGNED INT	NOT NULL	
index_count	The number of indexes in the table.	UN-SIGNED INT	NOT NULL	
indexcol_count	The total number of columns in all indexes in the table.	UN-SIGNED INT	NOT NULL	
map_handle	Used internally only.	UN-SIGNED INT	NOT NULL	
table_name	The name of the table.	VAR-CHAR (128)	NOT NULL	

Column name	Description	Type	Data constraint	Integrity constraints
object_id	A unique identifier for that table.	UNSIGNED INT	NOT NULL	Primary key.
sync_type	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.	VARCHAR (128)	NOT NULL	
table_name	The name of the table.	VARCHAR (128)	NOT NULL	
table_type	One of either sys for system tables or user for regular tables.	TINYINT	NOT NULL	
tpd_handle	Internal user only.	UNSIGNED INT	NOT NULL	

syscolumn system table

Description

Each row in the syscolumn system table describes one column.

Columns

Column name	Description	Type	Data constraint	Integrity constraints
column_name	A unique identifier of the column.	VARCHAR (128)	NOT NULL	
default	The default value for this column. For example, autoincrement.	VARCHAR (128)		
domain	The column domain, which is an enumerated value indicating the domain of the column.	UNSIGNED INT	NOT NULL	
domain_info	Used with a variable sized domain.	SMALLINT	NOT NULL	
nulls	Determines if the column allows nulls default.	CHAR(1)	NOT NULL	

Column name	Description	Type	Data constraint	Integrity constraints
object_id	A unique identifier for that column.	UNSIGNED INT	NOT NULL	Primary key.
table_id	The identifier of the table to which the column belongs.	UNSIGNED INT	NOT NULL	Primary key. Foreign key references object_id in sys-table.

sysindex system table

Description

Each row in the sysindex system table describes one index in the database.

Columns

Column name	Description	Type	Data constraint	Integrity constraints
check_on_commit	Indicates when referential integrity is checked to ensure there is a matching primary row for every foreign key. It is only required if type is foreign .	UNSIGNED INT		
index_name	A unique identifier for the index.	UNSIGNED INT	NOT NULL	Primary key. Foreign key references sysindex.
ixcol_count	The number of columns in the index.	UNSIGNED INT	NOT NULL	
nullable	Only required if type is foreign . Indicates if nulls are allowed.	BIT		
object_id	A unique identifier for an index.	UNSIGNED INT	NOT NULL	
primary_index_id	Only required if type is foreign . Lists the identifier of the primary index.	UNSIGNED INT		

Column name	Description	Type	Data constraint	Integrity constraints
primary_table_id	Only required if type is foreign . Lists the identifier of the primary table.	UNSIGNED INT		
root_handle	For internal use only.	UNSIGNED INT	NOT NULL	
table_id	A unique identifier for the table to which the index applies.	UNSIGNED INT	NOT NULL	Foreign key references systable.
type	The type of index. Can be one of: <ul style="list-style-type: none"> ◆ primary ◆ foreign ◆ key ◆ unique ◆ index 	SMALLINT (10)	NOT NULL	
hash_size	Stores the hash size used for index hashing.	SHORT		

See also

- ◆ [“sysixcol system table” on page 195](#)

sysixcol system table**Description**

Each row in the sysixcol system table describes one column of an index listed in sysindex.

Columns

Column name	Description	Type	Data constraint	Integrity constraints
column_id	A unique identifier for the column being indexed.	UNSIGNED INT	NOT NULL	Foreign key references syscolumn.
index_id	A unique identifier for the index that this index-column belongs to.	UNSIGNED INT	NOT NULL	Primary key. Foreign key references sysindex.

Column name	Description	Type	Data constraint	Integrity constraints
order	Indicates whether the column in the index is kept in ascending (A) or descending(D) order.	CHAR(1)	NOT NULL	
sequence	The order of the column in the index.	SMALLINT	NOT NULL	Primary key.
table_id	A unique identifier for the table to which the index applies.	UNSIGNED INT	NOT NULL	Primary key.

See also

- ◆ [“sysindex system table” on page 194](#)

syspublication system table**Description**

Each row in the syspublication system table describes a publication.

Columns

Column name	Description	Type	Data constraint	Integrity constraints
download_timestamp	The time of the last download.	UNSIGNED INT	NOT NULL	
last_sync	Used to keep track of upload progress.	UNSIGNED BIGINT	NOT NULL	
publication_id	A unique identifier for the publication.	UNSIGNED INT	NOT NULL	Primary key.
publication_name	The name of the publication.	CHAR(128)	NOT NULL	

See also

- ◆ [“sysarticle system table” on page 196](#)

sysarticle system table**Description**

Each row in the sysarticle system table describes a table that belongs to a publication.

Columns

Column name	Description	Type	Data constraint	Integrity constraints
publication_id	An identifier for the publication that this article belongs to.	UNSIGNED INT	NOT NULL	Primary key. Foreign key references syspublication.
table_id	The identifier of the table that belongs to the publication.	UNSIGNED INT	NOT NULL	Primary key. Foreign key references object_id in systable.
where_expr	An optional predicate to filter rows.	TINY INT		

See also

- ◆ [“syspublication system table” on page 196](#)

sysuldata system table**Description**

Each row in the sysuldata system table names value pairs of options and properties. For a list of available database options and properties, see [“UltraLite Database Settings Reference” on page 93](#).

Columns

Column name	Description	Type	Data constraint	Integrity constraints
long_setting	A BLOB for long values.	LONGBINARY		
name	The name of the property.	VARCHAR(128)	NOT NULL	Primary key.
setting	The value of the property.	VARCHAR(128)		
type	One of either sys for internals, opt for options or prop for properties	VARCHAR(128)	NOT NULL	Primary key.

Part IV. UltraLite SQL Reference

This part provides a reference for UltraLite SQL. UltraLite SQL is a unique subset of the SQL supported by SQL Anywhere databases.

CHAPTER 10

UltraLite SQL Elements Reference

Contents

Keywords in UltraLite	202
Identifiers in UltraLite	203
Strings in UltraLite	204
Comments in UltraLite	205
Numbers in UltraLite	206
The NULL value in UltraLite	207
Special values in UltraLite	208
Dates and times in UltraLite	211
Data types in UltraLite	212
Expressions in UltraLite	215
Operators in UltraLite	228
Variables in UltraLite	231
Query access plans in UltraLite	232

About this chapter

This chapter provides a complete reference for the SQL language elements used by UltraLite.

Keywords in UltraLite

Each SQL statement contains one or more keywords. SQL is case insensitive to keywords, but throughout these manuals, keywords are indicated in upper case. Some keywords cannot be used as identifiers without surrounding them in double quotes. These are called **reserved words**.

☞ For more information on reserved words, see “[Reserved words](#)” [*SQL Anywhere Server - SQL Reference*]. Note that UltraLite only supports a subset of those described. However, to be safe, always assume that all the words documented for SQL Anywhere apply as UltraLite reserved words as well.

Identifiers in UltraLite

Identifiers are names of objects in the database, such as user IDs, tables, and columns. UltraLite supports the same rules for identifiers as SQL Anywhere.

Note

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.

 For more information on identifiers, see “[Identifiers](#)” [*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” \[SQL Anywhere Server - SQL Reference\]](#)
- ◆ [“Character considerations” on page 31](#)

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 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 as a way to add comments to a SQL block.

Examples

- ◆ The following example illustrates the use of double-dash comments:

```
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 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:

```
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 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. */
```

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.

☞ For more information about NULL values, see [“NULL 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

Description

CURRENT DATE returns the current year, month, and day.

Data type

DATE

Usage

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 215](#)
- ◆ [“GETDATE function \[Date and time\]” on page 267](#)
- ◆ [“NOW function \[Date and time\]” on page 288](#)

CURRENT TIME special value

Description

The current hour, minute, second, and fraction of a second.

Data type

TIME

Usage

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 215](#)
- ◆ [“GETDATE function \[Date and time\]” on page 267](#)
- ◆ [“NOW function \[Date and time\]” on page 288](#)

CURRENT TIMESTAMP special value

Description

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

Usage

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 208](#)
- ◆ [“Expressions in UltraLite” on page 215](#)
- ◆ [“NOW function \[Date and time\]” on page 288](#)
- ◆ [“GETDATE function \[Date and time\]” on page 267](#)
- ◆ [“NOW function \[Date and time\]” on page 288](#)

SQLCODE special value

Description

Current SQLCODE value at the time the special value was evaluated.

Data type

String.

Usage

The SQLCODE value is set after each statement. You can check the SQLCODE to see whether or not the statement succeeded.

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.`

See also

- ◆ [“Expressions in UltraLite” on page 215](#)
- ◆ [SQL Anywhere 10 - Error Messages \[*SQL Anywhere 10 - Error Messages*\]](#).

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 [“Date parts” on page 237](#).

Data types in UltraLite

Available data types in UltraLite SQL include:

- ◆ Integer data types
- ◆ Decimal data types
- ◆ Floating-point data types
- ◆ Character data types
- ◆ Binary data types
- ◆ Date/time data types

Note

Domains (user-defined data types) are not supported in UltraLite SQL.

You can create a host variable with any one of the supported types. UltraLite supports a subset of the data types available in SQL Anywhere.

The following are the SQL data types supported in UltraLite databases.

Data type	Description
BIT	Boolean values (0 or 1). See “ BIT data type ” [SQL Anywhere Server - SQL Reference].
{ CHAR CHARACTER } (<i>max-length</i>)	Character data of <i>max-length</i> , in the range of 1-32767 bytes. When evaluating expressions, the maximum length for a temporary character value is 2048 bytes. See “ CHAR data type ” [SQL Anywhere Server - SQL Reference].
VARCHAR (<i>max-length</i>)	VARCHAR is used for variable-length character data of <i>max-length</i> . See “ VARCHAR data type ” [SQL Anywhere Server - SQL Reference].
LONG VARCHAR	Arbitrary length character data. Conditions in SQL statements (such as in the WHERE clause) cannot operate on LONG VARCHAR columns. The only operations allowed on LONG VARCHAR columns are to insert, update, or delete them, or to include them in the <i>select-list</i> of a query. You can cast strings to/from LONGVARCHAR data. See “ LONG VARCHAR data type ” [SQL Anywhere Server - SQL Reference].

Data type	Description
[UNSIGNED] BIGINT	An integer requiring 8 bytes of storage. See “ BIGINT data type ” [SQL Anywhere Server - SQL Reference].
{ DECIMAL DEC NUMERIC } (<i>precision</i> , <i>scale</i>)]	The representation of a decimal number using two parts: <i>precision</i> (total digits) and <i>scale</i> (digits that follow a decimal point). See “ DECIMAL data type ” [SQL Anywhere Server - SQL Reference], “ NUMERIC data type ” [SQL Anywhere Server - SQL Reference] and “ Decimal point position considerations ” on page 36.
DOUBLE [PRECISION]	A double-precision floating-point number. In this data type PRECISION is an optional part of the DOUBLE data type name. See “ DOUBLE data type ” [SQL Anywhere Server - SQL Reference].
FLOAT [(<i>precision</i>)]	A floating-point number, which may be single or double precision. See “ FLOAT data type ” [SQL Anywhere Server - SQL Reference].
[UNSIGNED] { INT INTEGER }	An unsigned integer requiring 4 bytes of storage. See “ INTEGER data type ” [SQL Anywhere Server - SQL Reference].
REAL	A single-precision floating-point number stored in 4 bytes. See “ REAL data type ” [SQL Anywhere Server - SQL Reference].
[UNSIGNED] SMALLINT	An integer requiring 2 bytes of storage. See “ SMALLINT data type ” [SQL Anywhere Server - SQL Reference].
[UNSIGNED] TINYINT	An integer requiring 1 byte of storage. See “ TINYINT data type ” [SQL Anywhere Server - SQL Reference].
DATE	A calendar date, such as a year, month, and day. See “ DATE data type ” [SQL Anywhere Server - SQL Reference].
TIME	The time of day, containing hour, minute, second, and fraction of a second. See “ TIME data type ” [SQL Anywhere Server - SQL Reference].
DATETIME	Identical to TIMESTAMP . See “ DATETIME data type ” [SQL Anywhere Server - SQL Reference].
TIMESTAMP	A point in time, containing year, month, day, hour, minute, second, and fraction of a second. See “ TIMESTAMP data type ” [SQL Anywhere Server - SQL Reference].
VARBINARY (<i>max-length</i>)	Identical to BINARY . See “ VARBINARY data type ” [SQL Anywhere Server - SQL Reference].
BINARY (<i>max-length</i>)	Binary data of maximum length <i>max-length</i> bytes. The maximum length should not exceed 2048 bytes. See “ BINARY data type ” [SQL Anywhere Server - SQL Reference].

Data type	Description
LONG BINARY	<p>Arbitrary length binary data. Conditions in SQL statements (such as in the WHERE clause) cannot operate on LONG BINARY columns. The only operations allowed on LONG BINARY columns are to insert, update, or delete them, or to include them in the <i>select-list</i> of a query.</p> <p>You can cast values to/from LONGBINARY data.</p> <p>See “LONG BINARY data type” [<i>SQL Anywhere Server - SQL Reference</i>].</p>
UNIQUEIDENTIFIER	<p>Typically used for a primary key or other unique column to hold UUID (Universally Unique Identifier) values that uniquely identify rows. UltraLite provides functions that generate 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 in this way can therefore be used as keys in a synchronization environment.</p> <p>See “UNIQUEIDENTIFIER data type” [<i>SQL Anywhere Server - SQL Reference</i>].</p>

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:

SQL Anywhere data type	UltraLite equivalent
MONEY	NUMERIC(19,4)
SMALLMONEY	NUMERIC(10,4)
TEXT	LONG VARCHAR
XML	LONG VARCHAR

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 216
- ◆ “Special values in UltraLite” on page 208
- ◆ “Column names in expressions” on page 216
- ◆ “UltraLite SQL Function Reference SQL Functions” on page 235
- ◆ “Subqueries in expressions” on page 219
- ◆ “Search conditions in UltraLite” on page 221
- ◆ “Data types in UltraLite” on page 212
- ◆ “CASE expressions” on page 217
- ◆ “Input parameters” on page 220

Constants in expressions

Description

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 ("").

Example

To use a possessive phrase, type the string literal as follows:

```
'John"s database'
```

See also

- ◆ “Escape sequences” [[SQL Anywhere Server - SQL Reference](#)]

Column names in expressions

Description

A column name is an identifier in an expression.

Syntax

`correlation-name.column-name`

Usage

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” on page 344](#)

IF expressions**Description**

Sets a search condition to return a specific subset of data.

Syntax

```

IF search-condition
THEN expression1
[ ELSE expression2 ]
ENDIF

```

Usage

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.

☞ For more information about TRUE, FALSE and UNKNOWN conditions, see [“NULL value” \[SQL Anywhere Server - SQL Reference\]](#) and [“Search conditions” \[SQL Anywhere Server - SQL Reference\]](#).

CASE expressions**Description**

The CASE expression 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
```

Usage

Case expressions can be used anywhere a regular expression can be used.

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. It selects a row from the Product table where the name column does not have a value of Tee Shirt and the quantity is less than fifty.

```
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

Description

Performs an aggregate computation that the UltraLite runtime does not provide.

Syntax

SUM(*expression*)

Usage

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 be either all aggregate expressions or none of the expressions can be an aggregate expression. 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, SUM(salary) is an aggregate expression:

```
SELECT SUM( salary )
FROM employee
```

Subqueries in expressions

Description

A subquery is a SELECT statement that is nested inside another SELECT statement.

Syntax

A subquery is structured like a regular query.

Usage

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 with references to 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).

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 )
```

See also

- ◆ [“SELECT statement” on page 342](#)
- ◆ [“Using Subqueries” \[SQL Anywhere Server - SQL Usage\]](#)
- ◆ [“Search conditions in UltraLite” on page 221](#)

Input parameters

Usage

Input parameters act 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 ] }
```

Usage

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 CHARACTER(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.

Example

The following embedded SQL statement has two input parameters:

```
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.

See also

- ◆ “Using host variables” [[UltraLite - C and C++ Programming](#)]
- ◆ “Preparing statements” [[SQL Anywhere Server - Programming](#)]
- ◆ UltraLite C/C++: “Data manipulation: Insert, Delete, and Update” [[UltraLite - C and C++ Programming](#)]
- ◆ UltraLite.NET: “Data manipulation: INSERT, UPDATE, and DELETE” [[UltraLite - .NET Programming](#)]
- ◆ UltraLite for AppForge: “Data manipulation: INSERT, UPDATE, and DELETE” [[UltraLite - AppForge Programming](#)]
- ◆ UltraLite for M-Business: “Data manipulation: INSERT, UPDATE and DELETE” [[UltraLite - M-Business Anywhere Programming](#)]

Search conditions in UltraLite**Usage**

To specify a search condition 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

```

search-condition:
  expression compare expression
| expression IS [ NOT ] { NULL | TRUE | FALSE | UNKNOWN }
| expression [ NOT ] BETWEEN expression AND expression
| expression [ NOT ] IN ( expression, ... )
| expression [ NOT ] IN ( subquery )
| expression [ NOT ] { ANY | ALL } ( subquery )
| expression [ NOT ] EXISTS ( subquery )
| NOT search-condition
| search-condition AND search-condition
| search-condition OR search-condition
| ( search-condition )

```

Parameters

```

compare:
= | > | < | >= | <= | <> | != | !< | !>

```

Usage

In UltraLite, search conditions can appear in the:

- ◆ WHERE clause
- ◆ HAVING clause
- ◆ ON phrase
- ◆ SQL queries

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 TRUE, FALSE, or UNKNOWN. This is called **three-valued logic**. The result of a comparison is UNKNOWN if either value being compared is the NULL value. Search conditions are satisfied only if the result of the condition is TRUE.

The different types of search conditions supported by UltraLite include:

- ◆ ALL conditions
- ◆ ANY conditions
- ◆ BETWEEN conditions
- ◆ EXISTS conditions
- ◆ IN conditions

These conditions are discussed in separate sections that follow.

Note

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

See also

- ◆ [“Comparison operators” on page 222](#)
- ◆ [“Three-valued logic” \[SQL Anywhere Server - SQL Reference\]](#)
- ◆ [“Subqueries in expressions” on page 219](#)

Comparison operators

Description

Any operator that allows two or more expressions to be compared with in a search condition.

Syntax

expression operator expression

Parameters

Operator	Interpretation
=	equal to
[NOT] LIKE	a text comparison, possibly using regular expressions
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
!=	not equal to

Operator	Interpretation
<>	not equal to
!>	not greater than
!<	not less than

Usage

Comparing dates In comparing dates, < means earlier and > means later.

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.

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 (!>).

```
SELECT ID, Name, Quantity
FROM Products
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND NOT UnitPrice > 10
```

```
SELECT ID, Name, Quantity
FROM Products
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND UnitPrice !> 10
```

See also

- ◆ [“Logical operators” on page 223](#)
- ◆ [“Search conditions in UltraLite” on page 221](#)

Logical operators

Description

Logical operators can do 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 }

Usage

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 TRUE, FALSE, or UNKNOWN. This is called **three-valued logic**. The result of a comparison is UNKNOWN if either value being compared is the NULL value. 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 OR *condition2*

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

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.

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

See also

- ◆ [“Three-valued logic” \[SQL Anywhere Server - SQL Reference\]](#)
- ◆ [“Comparison operators” on page 222](#)
- ◆ [“Search conditions in UltraLite” on page 221](#)

ALL conditions

Description

Use the ALL condition in conjunction with a comparison operators to compare a single value to the data values produced by the subquery.

Syntax

expression compare [NOT] ALL (*subquery*)

Parameters

compare:

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

Usage

UltraLite uses the specified comparison operator to compare the test value to each data value in the result set. If all of the comparisons yield TRUE results, the ALL test returns TRUE.

Example

Find the order and customer IDs of those orders placed after all products of order #2001 were shipped.

```
SELECT ID, CustomerID
FROM SalesOrders
WHERE OrderDate > ALL (
  SELECT ShipDate
  FROM SalesOrderItems
  WHERE ID=2001)
```

See also

- ◆ [“The ALL test” \[SQL Anywhere Server - SQL Usage\]](#)
- ◆ [“Comparison operators” on page 222](#)

ANY conditions

Description

Use the ANY condition in conjunction with a 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:

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

Usage

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.

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)
```

See also

- ◆ “The ANY test” [[SQL Anywhere Server - SQL Usage](#)]
- ◆ “Comparison operators” on page 222

BETWEEN conditions

Description

Specifies an inclusive range, in which the lower value and the upper value are searched for as well as the values they delimit.

Syntax

```
expression [ NOT ] BETWEEN start-expression AND end-expression
```

Usage

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 cheaper than \$10 or more expensive than \$15.

```
SELECT Name, UnitPrice
FROM Products
WHERE UnitPrice NOT BETWEEN 10 AND 15
```

EXISTS conditions

Description

Checks whether a subquery produces any rows of query results

Syntax

```
[ NOT ] EXISTS ( subquery )
```

Usage

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 conditions

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.

Usage

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 are from the following Canadian provinces: Ontario, Manitoba, and Quebec.

```
SELECT CompanyName , Province
FROM Customers
WHERE State IN( 'ON', 'MB', 'PQ' )
```

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.

See also

- ◆ [“Operator precedence” on page 229](#)
- ◆ [“Comparison operators” on page 222](#)
- ◆ [“Arithmetic operators” on page 228](#)
- ◆ [“String operators” on page 229](#)
- ◆ [“Bitwise operators” on page 229](#)
- ◆ [“Logical operators” on page 223](#)

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.

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
```

whereas 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.

Operator	Description
&	bitwise AND
	bitwise OR
^	bitwise exclusive OR
~	bitwise NOT

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.

```
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. ||
9. comparisons: >, <, <>, !=, <=, >=, [NOT] BETWEEN, [NOT] IN, [NOT] LIKE
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 except for the @@identity global variable.

@@identity

The @@identity global variable is the only SQL variable that you can with UltraLite. UltraLite uses this variable through API calls, however, not SQL queries.

☞ For more information about the @@identity global variable, see “@@identity global variable” [*SQL Anywhere Server - SQL Reference*].

See also

- ◆ “Global database ID considerations” on page 40
- ◆ “global_id option” on page 104

Query access plans in UltraLite

UltraLite query access plans show how tables and indexes are accessed when a query is executed. UltraLite includes a **query optimizer**: 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.

Performance tip

If you are not going to update data with the query, you should try specifying the `FOR READ ONLY` clause in your query. This clause may yield better performance.

When to view a query access plan

View a query access 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. See [“The temporary file” on page 16](#) for details.
- ◆ Which order tables are joined. This allows you to determine how performance is affected.
- ◆ Why a query is running slowly or to ensure that a query does not run slowly.

Viewing an UltraLite query access plan

As a development aid, you can use Interactive SQL to display an UltraLite plan that summarizes how a prepared statement is to be executed. The plan is displayed on a tab in the Results pane of the utility.

In UltraLite, a query 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.

Consider the following statement:


```
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 accomplished by reading all rows from the Invoice table (following index named primary) and then using the index named secondary from the Transactions table to read only the row whose inv_no column matches.

☞ For more information on Interactive SQL, see “[The Interactive SQL utility](#)” [*SQL Anywhere Server - Database Administration*]. For more information on abbreviations used in the plan, see “[Reading UltraLite access plans](#)” on page 233.

Reading UltraLite access 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.

Operation	Description
count (*)	Counts the number of rows in a table.
distinct [<i>plan</i>]	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.
dummy	No operation performed. It only occurs in two cases: <ul style="list-style-type: none"> ◆ When you specify DUMMY in a FROM clause. ◆ When the FROM clause is missing from the query.
filter [<i>plan</i>]	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.

Operation	Description
group-by [<i>plan</i>]	Creates an aggregate of GROUP BY results, in order to sort multiple rows of grouped data. Rows are listed in the order they occur and are grouped by comparing contiguous rows.
group-single [<i>plan</i>]	Creates an aggregate of GROUP BY results, but only when it is known that a single row will be returned.
keyset [<i>plan</i>]	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.
index-scan (<i>table-name</i> , <i>index-name</i>)	Reads only part of the table; the index is used to find the starting row.
join [<i>plan</i> , <i>plan</i>]	Performs an inner join between two plans.
lojoin [<i>plan</i> , <i>plan</i>]	Performs a left outer join between two plans.
like-scan (<i>table-name</i> , <i>index-name</i>)	Reads only part of a table; the index is used to find the starting row by pattern matching.
rowlimit [<i>plan</i>]	Performs the row limiting operation on propagated rows. Row limits are set by the TOP n or FIRST clause of the SELECT statement.
scan (<i>table-name</i> , <i>index-name</i>)	Reads an entire table following the order indicated by the index.
sub-query [<i>plan</i>]	Marks the start of a subquery.
temp [<i>plan</i>]	<p>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 accomplish this ordering.</p> <p>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 to which the index applies.</p>
union-all [<i>plan</i> , ..., <i>plan</i>]	Performs a UNION ALL operation on the rows generated in the underlying plan.

CHAPTER 11

UltraLite SQL Function Reference

Contents


Function types	236
Alphabetical list of functions	241

About this chapter

Functions are used to return information from the database. They are allowed anywhere an expression is allowed. The chapter includes a grouping of functions by type, followed by an alphabetical list of functions.

Before you begin

Functions use the same syntax conventions used by SQL statements. Ensure you understand these conventions and how they are used in this chapter.

 For a complete list of syntax conventions, see [“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.

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 \[Aggregate\]” on page 245](#)
- ◆ [“COUNT function \[Aggregate\]” on page 255](#)
- ◆ [“LIST function \[Aggregate\]” on page 277](#)
- ◆ [“MAX function \[Aggregate\]” on page 281](#)
- ◆ [“MIN function \[Aggregate\]” on page 282](#)
- ◆ [“SUM function \[Aggregate\]” on page 308](#)

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 \[Data type conversion\]” on page 247](#)
- ◆ [“CONVERT function \[Data type conversion\]” on page 252](#)
- ◆ [“HEXTOINT function \[Data type conversion\]” on page 269](#)
- ◆ [“INTTOHEX function \[Data type conversion\]” on page 273](#)
- ◆ [“ISDATE function \[Data type conversion\]” on page 273](#)

UltraLite Date and time functions

Date and time functions perform operations on date and time data types or return date or time information.

In this chapter, the term **datetime** is used to mean date or time or timestamp. The specific data type DATETIME is indicated as DATETIME.

☞ For more information on datetime data types, see [“Data types in UltraLite” on page 212](#).

List of functions

The following date and time functions are available:

- ◆ “DATE function [Date and time]” on page 257
- ◆ “DATEADD function [Date and time]” on page 257
- ◆ “DATEDIFF function [Date and time]” on page 258
- ◆ “DATEFORMAT function [Date and time]” on page 259
- ◆ “DATENAME function [Date and time]” on page 260
- ◆ “DATEPART function [Date and time]” on page 260
- ◆ “DATETIME function [Date and time]” on page 261
- ◆ “DAY function [Date and time]” on page 262
- ◆ “DAYNAME function [Date and time]” on page 262
- ◆ “DAYS function [Date and time]” on page 263
- ◆ “DOW function [Date and time]” on page 265
- ◆ “GETDATE function [Date and time]” on page 267
- ◆ “HOUR function [Date and time]” on page 269
- ◆ “HOURS function [Date and time]” on page 270
- ◆ “MINUTE function [Date and time]” on page 282
- ◆ “MINUTES function [Date and time]” on page 283
- ◆ “MONTH function [Date and time]” on page 285
- ◆ “MONTHNAME function [Date and time]” on page 285
- ◆ “MONTHS function [Date and time]” on page 286
- ◆ “NOW function [Date and time]” on page 288
- ◆ “QUARTER function [Date and time]” on page 291
- ◆ “SECOND function [Date and time]” on page 297
- ◆ “SECONDS function [Date and time]” on page 298
- ◆ “TODAY function [Date and time]” on page 310
- ◆ “WEEKS function [Date and time]” on page 314
- ◆ “YEAR function [Date and time]” on page 315
- ◆ “YEARS function [Date and time]” on page 315
- ◆ “YMD function [Date and time]” on page 316

Date parts

Many of the date functions use dates built from **date parts**. The following table displays allowed values of date parts.

Date part	Abbreviation	Values
Year	yy	1–9999
Quarter	qq	1–4

Date part	Abbreviation	Values
Month	mm	1–12
Week	wk	1–54. Weeks begin on Sunday.
Day	dd	1–31
Dayofyear	dy	1–366
Weekday	dw	1–7 (Sunday = 1, ..., Saturday = 7)
Hour	hh	0–23
Minute	mi	0–59
Second	ss	0–59
Millisecond	ms	0–999
Calyearofweek	cyr	Integer. 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.
Calweekofyear	cwk	1–54. The week number within the year that contains the specified date.
Caldayofweek	cdw	1–7. (Monday = 1, ..., Sunday = 7)

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 \[Miscellaneous\]” on page 242](#)
- ◆ [“COALESCE function \[Miscellaneous\]” on page 251](#)
- ◆ [“EXPLANATION function \[Miscellaneous\]” on page 266](#)
- ◆ [“GREATER function \[Miscellaneous\]” on page 268](#)
- ◆ [“IFNULL function \[Miscellaneous\]” on page 271](#)
- ◆ [“ISNULL function \[Miscellaneous\]” on page 273](#)
- ◆ [“LESSER function \[Miscellaneous\]” on page 276](#)
- ◆ [“NEWID function \[Miscellaneous\]” on page 287](#)
- ◆ [“NULLIF function \[Miscellaneous\]” on page 288](#)

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 \[Numeric\]” on page 241](#)
- ◆ [“ACOS function \[Numeric\]” on page 241](#)
- ◆ [“ASIN function \[Numeric\]” on page 243](#)
- ◆ [“ATAN function \[Numeric\]” on page 244](#)
- ◆ [“ATAN2 function \[Numeric\]” on page 244](#)
- ◆ [“CEILING function \[Numeric\]” on page 248](#)
- ◆ [“COS function \[Numeric\]” on page 254](#)
- ◆ [“COT function \[Numeric\]” on page 255](#)
- ◆ [“DEGREES function \[Numeric\]” on page 264](#)
- ◆ [“EXP function \[Numeric\]” on page 266](#)
- ◆ [“FLOOR function \[Numeric\]” on page 267](#)
- ◆ [“LOG function \[Numeric\]” on page 278](#)
- ◆ [“LOG10 function \[Numeric\]” on page 279](#)
- ◆ [“MOD function \[Numeric\]” on page 284](#)
- ◆ [“PI function \[Numeric\]” on page 290](#)
- ◆ [“POWER function \[Numeric\]” on page 291](#)
- ◆ [“RADIANS function \[Numeric\]” on page 292](#)
- ◆ [“REMAINDER function \[Numeric\]” on page 292](#)
- ◆ [“ROUND function \[Numeric\]” on page 296](#)
- ◆ [“SIGN function \[Numeric\]” on page 300](#)
- ◆ [“SIN function \[Numeric\]” on page 301](#)
- ◆ [“SQRT function \[Numeric\]” on page 303](#)
- ◆ [“TAN function \[Numeric\]” on page 309](#)
- ◆ [“TRUNCNUM function \[Numeric\]” on page 311](#)

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 \[String\]” on page 242](#)
- ◆ [“BYTE_LENGTH function \[String\]” on page 246](#)
- ◆ [“BYTE_SUBSTR function \[String\]” on page 246](#)
- ◆ [“CHAR function \[String\]” on page 249](#)

- ◆ “CHARINDEX function [String]” on page 249
- ◆ “CHAR_LENGTH function [String]” on page 250
- ◆ “DIFFERENCE function [String]” on page 264
- ◆ “INSERTSTR function [String]” on page 272
- ◆ “LCASE function [String]” on page 274
- ◆ “LEFT function [String]” on page 275
- ◆ “LENGTH function [String]” on page 275
- ◆ “LOCATE function [String]” on page 278
- ◆ “LOWER function [String]” on page 280
- ◆ “LTRIM function [String]” on page 280
- ◆ “PATINDEX function [String]” on page 289
- ◆ “REPEAT function [String]” on page 293
- ◆ “REPLACE function [String]” on page 294
- ◆ “REPLICATE function [String]” on page 295
- ◆ “RIGHT function [String]” on page 295
- ◆ “RTRIM function [String]” on page 297
- ◆ “SIMILAR function [String]” on page 300
- ◆ “SOUNDEX function [String]” on page 302
- ◆ “SPACE function [String]” on page 303
- ◆ “STR function [String]” on page 304
- ◆ “STRING function [String]” on page 305
- ◆ “STRTOUUID function [String]” on page 305
- ◆ “STUFF function [String]” on page 306
- ◆ “SUBSTRING function [String]” on page 307
- ◆ “TRIM function [String]” on page 310
- ◆ “UCASE function [String]” on page 311
- ◆ “UPPER function [String]” on page 312
- ◆ “UUIDTOSTR function [String]” on page 313

Alphabetical list of 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 236](#).

ABS function [Numeric]

Description

Returns the absolute value of a numeric expression.

Syntax

ABS(*numeric-expression*)

Parameters

numeric expression The number whose absolute value is to be returned.

Standards and compatibility

◆ **SQL/2003** SQL foundation feature outside of core SQL.

Example

The following statement returns the value 66.

```
SELECT ABS( -66 );
```

ACOS function [Numeric]

Description

Returns the arc-cosine, in radians, of a numeric expression.

Syntax

ACOS(*numeric-expression*)

Parameters

numeric-expression The cosine of the angle.

Usage

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result.

See also

- ◆ [“ASIN function \[Numeric\]” on page 243](#)
- ◆ [“ATAN function \[Numeric\]” on page 244](#)
- ◆ [“ATAN2 function \[Numeric\]” on page 244](#)

- ◆ [“COS function \[Numeric\]” on page 254](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the arc-cosine value for 0.52.

```
SELECT ACOS( 0.52 );
```

ARGN function [Miscellaneous]

Description

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.

Usage

Using the value of the *integer-expression* as n, returns the nth argument (starting at 1) from the remaining list of arguments. While the expressions can be of any data type, they must all be of the same data type. The integer expression must be from one to the number of expressions in the list or NULL is returned. Multiple expressions are separated by a comma.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 6.

```
SELECT ARGN( 6, 1,2,3,4,5,6 );
```

ASCII function [String]

Description

Returns the integer ASCII value of the first byte in a string-expression.

Syntax

```
ASCII(string-expression )
```

Parameters

string-expression The string.

Usage

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

- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 90.

```
SELECT ASCII( 'Z' );
```

ASIN function [Numeric]

Description

Returns the arc-sine, in radians, of a number.

Syntax

ASIN(*numeric-expression*)

Parameters

numeric-expression The sine of the angle.

Usage

The SIN and ASIN functions are inverse operations.

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result.

See also

- ◆ [“ACOS function \[Numeric\]” on page 241](#)
- ◆ [“ATAN function \[Numeric\]” on page 244](#)
- ◆ [“ATAN2 function \[Numeric\]” on page 244](#)
- ◆ [“SIN function \[Numeric\]” on page 301](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the arc-sine value for 0.52.

```
SELECT ASIN( 0.52 );
```

ATAN function [Numeric]

Description

Returns the arc-tangent, in radians, of a number.

Syntax

```
ATAN( numeric-expression )
```

Usage

The ATAN and TAN functions are inverse operations.

Parameters

numeric-expression The tangent of the angle.

Usage

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result.

See also

- ◆ [“ACOS function \[Numeric\]” on page 241](#)
- ◆ [“ASIN function \[Numeric\]” on page 243](#)
- ◆ [“ATAN2 function \[Numeric\]” on page 244](#)
- ◆ [“TAN function \[Numeric\]” on page 309](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the arc-tangent value for 0.52.

```
SELECT ATAN( 0.52 );
```

ATAN2 function [Numeric]

Description

Returns the arc-tangent, in radians, of the ratio of two numbers.

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.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result.

See also

- ◆ [“ACOS function \[Numeric\]” on page 241](#)
- ◆ [“ASIN function \[Numeric\]” on page 243](#)
- ◆ [“ATAN function \[Numeric\]” on page 244](#)
- ◆ [“TAN function \[Numeric\]” on page 309](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the arc-tangent value for the ratio 0.52 to 0.60.

```
SELECT ATAN2( 0.52, 0.60 );
```

AVG function [Aggregate]

Description

Computes the average, for a set of rows, of a numeric expression or of a set unique values.

Syntax 1

```
AVG( numeric-expression | DISTINCT numeric-expression )
```

Parameters

- numeric-expression** The expression whose average is calculated over a set of rows.
- DISTINCT numeric-expression** Computes the average of the unique numeric values in the input.

Usage

This average does not include rows where the *numeric-expression* is the NULL value. Returns the NULL value for a group containing no rows.

See also

- ◆ [“SUM function \[Aggregate\]” on page 308](#)
- ◆ [“COUNT function \[Aggregate\]” on page 255](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature. Syntax 2 is feature T611.

Example

The following statement returns the value 49988.623200.

```
SELECT AVG( Salary ) FROM Employees ;
```

The following statement could be used to determine the average based on unique prices in the production list:

```
SELECT AVG( DISTINCT ListPrice ) FROM Production ;
```

BYTE_LENGTH function [String]

Description

Returns the number of bytes in a string.

Syntax

```
BYTE_LENGTH( string-expression )
```

Parameters

string-expression The string whose length is to be calculated.

Usage

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 250](#)
- ◆ [“DATALENGTH function \[System\]” on page 256](#)
- ◆ [“LENGTH function \[String\]” on page 275](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 12.

```
SELECT BYTE_LENGTH( 'Test Message' );
```

BYTE_SUBSTR function [String]

Description

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.

Usage

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 307](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value Test.

```
SELECT BYTE_SUBSTR( 'Test Message', 1, 4 );
```

CAST function [Data type conversion]

Description

Returns the value of an expression converted to a supplied data type.

Syntax

CAST(*expression AS data type*)

Parameters

expression The expression to be converted.

data type The target data type.

Usage

If you do not indicate a length for character string types, the database server chooses an appropriate length. If neither precision nor scale is specified for a DECIMAL conversion, the database server selects appropriate values.

See also

- ◆ [“CONVERT function \[Data type conversion\]” on page 252](#)
- ◆ [“LEFT function \[String\]” on page 275](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature.

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) );
```

CEILING function [Numeric]

Description

Returns the ceiling of a number.

Syntax

```
CEILING( numeric-expression )
```

Parameters

numeric-expression The number whose ceiling is to be calculated.

Usage

The Ceiling function returns the first integer that is greater or equal to a given value. For positive numbers, this is also known as "rounding up."

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result.

See also

- ◆ [“FLOOR function \[Numeric\]” on page 267](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 60.

```
SELECT CEILING( 59.84567 );
```

CHAR function [String]

Description

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.

Usage

The character returned corresponds to the supplied numeric expression in the current database character set, according to a binary sort order.

CHAR returns NULL for integer expressions with values greater than 255 or less than zero.

See also

- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value Y.

```
SELECT CHAR( 89 );
```

CHARINDEX function [String]

Description

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.

Usage

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.

See also

- ◆ [“SUBSTRING function \[String\]” on page 307](#)
- ◆ [“REPLACE function \[String\]” on page 294](#)
- ◆ [“LOCATE function \[String\]” on page 278](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns last and first names from the Surname and GivenName tables, but only when the last name includes the letter K:

```
SELECT Surname, GivenName
FROM Employees
WHERE CHARINDEX( 'K', Surname ) = 1 ;
```

Results returned:

Surname	GivenName
Klobucher	James
Kuo	Felicia
Kelly	Moira

CHAR_LENGTH function [String]**Description**

Returns the number of characters in a string.

Syntax

```
CHAR_LENGTH ( string-expression )
```

Parameters

- string-expression** The string whose length is to be calculated.

Usage

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 246](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature.

Example

The following statement returns the value 8.

```
SELECT CHAR_LENGTH( 'Chemical' );
```

COALESCE function [Miscellaneous]

Description

Returns the first non-NULL expression from a list. This function is identical to the ISNULL function.

Syntax

```
COALESCE( expression, expression [ , ... ] )
```

Parameters

expression Any expression.

At least two expressions must be passed into the function.

Usage

The result is NULL only if all the arguments are NULL.

The parameters can be of any scalar type, but not necessarily same type.

Standards and compatibility

- ◆ **SQL/2003** Core feature.

Example

The following statement returns the value 34.

```
SELECT COALESCE( NULL, 34, 13, 0 );
```

CONVERT function [Data type conversion]**Description**

Returns an expression converted to a supplied data type.

Syntax

```
CONVERT( data type, expression [ , format-style ] )
```

Parameters

data type The data type to which the expression is converted.

expression The expression to be converted.

format-style The style code to apply to the outputted 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).

Without century (yy) style codes	With century (yyyy) style codes	Output format
-	0 or 100	Mmm dd yyyy hh:nnAA
1	101	mm/dd/yy[yy]
2	102	[yy]yy.mm.dd
3	103	dd/mm/yy[yy]
4	104	dd.mm.yy[yy]
5	105	dd-mm-yy[yy]
6	106	dd Mmm yy[yy]
7	107	Mmm dd, yy[yy]
8	108	hh:nn:ss
-	9 or 109	Mmm dd yyyy hh:nn:ss:sssAA
10	110	mm-dd-yy[yy]
11	111	[yy]yy/mm/dd
12	112	[yy]yyymmdd

Without century (yy) style codes	With century (yyyy) style codes	Output format
-	13 or 113	dd Mmm yyyy hh:nn:ss:sss (24 hour clock, Europe default + milliseconds, 4-digit year)
-	14 or 114	hh:nn:ss:sss (24 hour clock)
-	20 or 120	yyyy-mm-dd hh:nn:ss (24-hour clock, ODBC canonical, 4-digit year)
-	21 or 121	yyyy-mm-dd hh:nn:ss:sss (24 hour clock, ODBC canonical with milliseconds, 4-digit year)

Usage

If no *format-style* argument is provided, style code 0 is used.

☞ For a description of the styles produced by each output symbol (such as Mmm), see “[date_format property](#)” on page 97.

See also

- ◆ “[CAST function \[Data type conversion\]](#)” on page 247

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statements illustrate the use of format style.

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 104 ) FROM SalesOrders ;
```

OrderDate
16.03.2000
20.03.2000
23.03.2000
25.03.2000
...

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 7 ) FROM SalesOrders;
```

OrderDate
Mar 16, 00
Mar 20, 00
Mar 23, 00
Mar 25, 00
...

The following statement illustrates conversion to an integer, and returns the value 5.

```
SELECT CONVERT( integer, 5.2 );
```

COS function [Numeric]

Description

Converts a number from radians to cosine.

Syntax

```
COS( numeric-expression )
```

Parameters

numeric-expression The angle, in radians.

Usage

The COS function returns the cosine of the angle given by *numeric-expression*.

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

See also

- ◆ [“ACOS function \[Numeric\]” on page 241](#)
- ◆ [“COT function \[Numeric\]” on page 255](#)
- ◆ [“SIN function \[Numeric\]” on page 301](#)
- ◆ [“TAN function \[Numeric\]” on page 309](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value of the cosine of an angle 0.52 radians.

```
SELECT COS( 0.52 );
```

COT function [Numeric]

Description

Converts a number from radians to cotangent.

Syntax

COT(*numeric-expression*)

Parameters

numeric-expression The angle, in radians.

Usage

The COT function returns the cotangent of the angle given by *numeric-expression*.

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

See also

- ◆ [“COS function \[Numeric\]” on page 254](#)
- ◆ [“SIN function \[Numeric\]” on page 301](#)
- ◆ [“TAN function \[Numeric\]” on page 309](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the cotangent value of 0.52.

```
SELECT COT( 0.52 );
```

COUNT function [Aggregate]

Description

Counts the number of rows in a group depending on the specified parameters.

Syntax 1

```
COUNT(  
*  
| expression  
| DISTINCT expression  
)
```

Parameters

- * Returns the number of rows in each group.
- expression** Returns the number of rows of each group.

DISTINCT expression Returns the number of different values in the expression.

Usage

Rows where the value is the NULL value are not included in the count.

See also

- ◆ [“AVG function \[Aggregate\]” on page 245](#)
- ◆ [“SUM function \[Aggregate\]” on page 308](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature. Syntax 2 is feature T611.

Example

The following statement returns each unique city, and the number of rows with that city value.

```
SELECT City , COUNT(*) FROM Employees GROUP BY City;
```

DATALENGTH function [System]

Description

Returns the length, in bytes, of the underlying storage for the result of an expression.

Syntax

DATALENGTH(*expression*)

Parameters

expression *expression* is usually a column name. If *expression* is a string constant, you must enclose it in quotes.

Usage

The return values of the DATALENGTH function are as follows:

Data type	DATALENGTH
SMALLINT	2
INTEGER	4
DOUBLE	8
CHAR	Length of the data
BINARY	Length of the data

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example


The following statement returns the value 27, the longest string in the `CompanyName` column.

```
SELECT MAX( DATALENGTH( CompanyName ) )  
FROM Customers;
```

DATE function [Date and time]

Description

Converts the expression into a date, and removes any hours, minutes, or seconds.

 For information about controlling the interpretation of date formats, see [“date_order property” on page 100](#).

Syntax

```
DATE( expression )
```

Parameters

expression The value to be converted to date format. An expression is usually a string.

Standards and compatibility

◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 1999-01-02 as a date.

```
SELECT DATE( '1999-01-02 21:20:53' );
```

DATEADD function [Date and time]

Description

Returns the date produced by adding a number of the date parts to a date.

Syntax


```
DATEADD( date-part, numeric-expression, date-expression )
```

date-part :

year | **quarter** | **month** | **week** | **day** | **dayofyear** | **hour** | **minute** | **second** | **millisecond**

Parameters

date-part The date part to be added to the date.

 For more information about date parts, see [“Date parts” on page 237](#).

numeric-expression The number of date parts to be added to the date. The *numeric_expression* can be any numeric type, but the value is truncated to an integer.

date-expression The date to be modified.

Standards and compatibility

◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value: 1995-11-02 00:00:00.000.

```
SELECT DATEADD( month, 102, '1987/05/02' );
```

DATEDIFF function [Date and time]

Description

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**

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 [“Date parts” on page 237](#).

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.

Usage

This function calculates the number of date parts between two specified dates. The result is a signed integer value equal to (date2 – date1), 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.

For the smaller time units there are overflow values:

- ◆ **milliseconds** 24 days
- ◆ **seconds** 68 years
- ◆ **minutes** 4083 years
- ◆ **others** No overflow limit

The function returns an overflow error if you exceed these limits.

Standards and compatibility

- ◆ **SQL/2003** Transact-SQL extension.

Example

The following statement returns 1.

```
SELECT DATEDIFF( hour, '4:00AM', '5:50AM' );
```

The following statement returns 102.

```
SELECT DATEDIFF( month, '1987/05/02', '1995/11/15' );
```

The following statement returns 0.

```
SELECT DATEDIFF( day, '00:00', '23:59' );
```

The following statement returns 4.

```
SELECT DATEDIFF( day,  
                '1999/07/19 00:00',  
                '1999/07/23 23:59' );
```

The following statement returns 0.

```
SELECT DATEDIFF( month, '1999/07/19', '1999/07/23' );
```

The following statement returns 1.

```
SELECT DATEDIFF( month, '1999/07/19', '1999/08/23' );
```

DATEFORMAT function [Date and time]

Description

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 on date format descriptions, see “[date_format property](#)” on page 97.

Usage

Any allowable date format can be used for the string-expression.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value Jan 01, 1989.

```
SELECT DATEFORMAT( '1989-01-01', 'Mmm dd, yyyy' );
```

DATENAME function [Date and time]

Description

Returns the name of the specified part (such as the month June) of a datetime value, as a character string.

Syntax

```
DATENAME( date-part, date-expression )
```

Parameters

date-part The date part to be named.

☞ For a complete listing of allowed date parts, see “[Date parts](#)” on page 237.

date-expression The date for which the date part name is to be returned. The date must contain the requested *date-part*.

Usage

The DATENAME function returns a string, even if the result is numeric, such as 23 for the day.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value May.

```
SELECT DATENAME( month, '1987/05/02' );
```

DATEPART function [Date and time]

Description


Returns the value of part of a datetime value.

Syntax

DATEPART(*date-part*, *date-expression*)

Parameters

date-part The date part to be returned.

 For a complete listing of allowed date parts, see [“Date parts” on page 237](#).

date-expression The date for which the part is to be returned.

Usage

The date must contain the *date-part* field.

Standards and compatibility

◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 5.

```
SELECT DATEPART( month , '1987/05/02' );
```

DATETIME function [Date and time]

Description

Converts an expression into a timestamp.

Syntax

DATETIME(*expression*)

Parameters

expression The expression to be converted. It is generally a string.

Usage

Attempts to convert numerical values return an error.

Standards and compatibility

◆ **SQL/2003** Vendor extension.

Example

The following statement returns a timestamp with value 1998-09-09 12:12:12.000.

```
SELECT DATETIME( '1998-09-09 12:12:12.000' );
```

DAY function [Date and time]

Description

Returns an integer from 1 to 31.

Syntax

DAY(*date-expression*)

Parameters

date-expression The date.

Usage

The integers 1 to 31 correspond to the day of the month in a date.

Standards and compatibility

◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 12.

```
SELECT DAY( '2001-09-12' );
```

DAYNAME function [Date and time]

Description

Returns the name of the day of the week from a date.

Syntax

DAYNAME(*date-expression*)

Parameters

date-expression The date.

Usage

The English names are returned as: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday.

Standards and compatibility

◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value Saturday.

```
SELECT DAYNAME ( '1987/05/02' );
```

DAYS function [Date and time]

Description

A function that evaluates days. For specific details, see this function's usage.

Syntax 1: integer

`DAYS([datetime-expression,] datetime-expression)`

Syntax 2: timestamp

`DAYS(datetime-expression, integer-expression)`

Parameters

datetime-expression A date and time.

integer-expression The number of days to be added to the *datetime-expression*. If the *integer-expression* is negative, the appropriate number of days is subtracted from the timestamp. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a date or timestamp.

☞ For information on casting data types, see “[CAST function \[Data type conversion\]](#)” on page 247.

Usage

The behavior of this function can vary depending on what you supply:

- ◆ If you give a single date, this function returns the number of days since 0000-02-29.

Note

0000-02-29 is not meant to imply an actual date; it is the date used by the date algorithm.

- ◆ If you give two dates, this function returns the integer number of days between them. Instead, use the DATEDIFF function.
- ◆ If you give a date and an integer, this function adds the integer number of days to the specified date. Instead, use the DATEADD function.

This function ignores hours, minutes, and seconds.

See also

- ◆ “[DATEDIFF function \[Date and time\]](#)” on page 258
- ◆ “[DATEADD function \[Date and time\]](#)” on page 257

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the integer 729889.

```
SELECT DAYS( '1998-07-13 06:07:12' );
```

The following statements return the integer value -366 , indicating that the second date is 366 days prior to the first. It is recommended that you use the second example (DATEDIFF).

```
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 1999-07-14 00:00:00.000. It is recommended that you use the second example (DATEADD).

```
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 );  
  
SELECT DATEADD( day, 366, '1998-07-13' );
```

DEGREES function [Numeric]

Description

Converts a number from radians to degrees.

Syntax

```
DEGREES( numeric-expression )
```

Parameters

numeric-expression An angle in radians.

Usage

The DEGREES function returns the degrees of the angle given by *numeric-expression*.

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 29.79380534680281.

```
SELECT DEGREES( 0.52 );
```

DIFFERENCE function [String]

Description

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.

Usage

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 302](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 3.

```
SELECT DIFFERENCE( 'test', 'chest' );
```

DOW function [Date and time]

Description

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 date.

Usage

The DOW function is not affected by the value specified for the `first_day_of_week` database option. For example, even if `first_day_of_week` is set to Monday, the DOW function returns a 2 for Monday.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 5.

```
SELECT DOW( '1998-07-09' );
```

EXP function [Numeric]

Description

Returns the exponential function, e to the power of a number.

Syntax

EXP(*numeric-expression*)

Parameters

numeric-expression The exponent.

Usage

The EXP function returns the exponential of the value specified by *numeric-expression*.

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The statement returns the value 3269017.3724721107.

```
SELECT EXP( 15 );
```

EXPLANATION function [Miscellaneous]

Returns the plan optimization strategy of a SQL statement.

Syntax

EXPLANATION(*string-expression*)

Parameters

string-expression The SQL statement, which is commonly a SELECT statement, but can also be an UPDATE or DELETE statement.

Usage

The optimization is returned as a string.

This information can help you decide which indexes to add or how to structure your database for better performance.

In Interactive SQL, you can view the plan for any SQL statement on the Plan tab in the Results pane.

See also

- ◆ [“Query access plans in UltraLite” on page 232](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement passes a **SELECT** statement as a string parameter and returns the plan for executing the query.

```
SELECT EXPLANATION( 'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

FLOOR function [Numeric]

Description

Returns the floor of (largest integer not greater than) a number.

Syntax

```
FLOOR( numeric-expression )
```

Parameters

numeric- expression The number, usually a float.

Usage

This function converts its arguments to **DOUBLE**, performs the computation in double-precision floating point, and returns a **DOUBLE** as the result.

See also

- ◆ [“CEILING function \[Numeric\]” on page 248](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statements returns a Floor value of 123

```
SELECT FLOOR (123);
```

The following statements returns a Floor value of 123

```
SELECT FLOOR (123.45);
```

The following statements returns a Floor value of -124

```
SELECT FLOOR (-123.45);
```

GETDATE function [Date and time]

Description

Returns the current year, month, day, hour, minute, second and fraction of a second.

Syntax

GETDATE()

Usage

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 288](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the system date and time.

```
SELECT GETDATE( );
```

GREATER function [Miscellaneous]

Description

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.

Usage

If the parameters are equal, the first is returned.

See also

- ◆ [“LESSER function \[Miscellaneous\]” on page 276](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 10.

```
SELECT GREATER( 10, 5 ) FROM dummy;
```

HEXTOINT function [Data type conversion]

Description

Returns the decimal integer equivalent of a hexadecimal string.

Syntax

```
HEXTOINT( hexadecimal-string )
```

Parameters

hexadecimal-string The string to be converted to an integer.

Usage

The HEXTOINT function accepts string literals or variables consisting only of digits and the uppercase or lowercase letters A-F, with or without a 0x prefix. The following are all valid uses of HEXTOINT:

```
SELECT HEXTOINT( '0xFFFFFFFF' );
SELECT HEXTOINT( '0x00000100' );
SELECT HEXTOINT( '100' );
SELECT HEXTOINT( '0xffffffff80000001' );
```

The HEXTOINT function removes the 0x prefix, if present. If the data exceeds 8 digits, it must represent a value that can be represented as a signed 32-bit integer value.

The HEXTOINT function returns the platform-independent SQL INTEGER equivalent of the hexadecimal string. The hexadecimal value represents a negative integer if the 8th digit from the right is one of the digits 8–9 and the uppercase or lowercase letters A–F and the previous leading digits are all uppercase or lowercase letter F. The following is not a valid use of HEXTOINT since the argument represents a positive integer value that cannot be represented as a signed 32-bit integer:

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

See also

- ◆ [“INTTOHEX function \[Data type conversion\]” on page 273](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 420.

```
SELECT HEXTOINT( '1A4' );
```

HOURL function [Date and time]

Description

Returns the hour component of a datetime.

Syntax

HOUR(*datetime-expression*)

Parameters

datetime-expression The datetime.

Usage

The value returned is a number from 0 to 23 corresponding to the datetime hour.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 21:

```
SELECT HOUR( '1998-07-09 21:12:13' );
```

HOURS function [Date and time]

Description

A function that evaluates hours. For specific details, see this function's usage.

Syntax 1: integer

HOURS ([*datetime-expression*,] *datetime-expression*)

Syntax 2: timestamp

HOURS (*datetime-expression*, *integer-expression*)

Parameters

datetime-expression A date and time.

integer-expression The number of hours to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of hours is subtracted from the datetime. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a DATETIME data type.

☞ For information on casting data types, see [“CAST function \[Data type conversion\]” on page 247](#).

Usage

The behavior of this function can vary depending on what you supply:

- ◆ If you give a single date, this function returns the number of hours since 0000-02-29.

Note

0000-02-29 is not meant to imply an actual date; it is the date used by the date algorithm.

- ◆ If you give two timestamps, this function returns the integer number of hours between them. Instead, use the DATEDIFF function.
- ◆ If you give a date and an integer, this function adds the integer number of hours to the specified timestamp. Instead, use the DATEADD function.

See also

- ◆ [“DATEDIFF function \[Date and time\]” on page 258](#)
- ◆ [“DATEADD function \[Date and time\]” on page 257](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statements return the value 4, signifying that the second timestamp is four hours after the first. It is recommended that you use the second example (DATEDIFF).

```
SELECT HOURS( '1999-07-13 06:07:12',
             '1999-07-13 10:07:12' );

SELECT DATEDIFF( hour,
                '1999-07-13 06:07:12',
                '1999-07-13 10:07:12' );
```

The following statement returns the value 17517342.

```
SELECT HOURS( '1998-07-13 06:07:12' );
```

The following statements return the datetime 1999-05-13 02:05:07.000. It is recommended that you use the second example (DATEADD).

```
SELECT HOURS(
    CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 );

SELECT DATEADD( hour, 5, '1999-05-12 21:05:07' );
```

IFNULL function [Miscellaneous]

Description

Returns the first non NULL expression.

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.

Usage

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.

Standards and compatibility

- ◆ **SQL/2003** Transact-SQL extension.

Example

The following statement returns the value -66.

```
SELECT IFNULL( NULL, -66 );
```

The following statement returns NULL, because the first expression is not NULL and there is no third expression.

```
SELECT IFNULL( -66, -66 );
```

INSERTSTR function [String]

Description

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.

Usage

See also

- ◆ [“STUFF function \[String\]” on page 306](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value backoffice.

```
SELECT INSERTSTR( 0, 'office ', 'back' );
```


INTTOHEX function [Data type conversion]

Description

Returns a string containing the hexadecimal equivalent of an integer.

Syntax

```
INTTOHEX( integer-expression )
```

Parameters

integer-expression The integer to be converted to hexadecimal.

See also

- ◆ [“HEXTOINT function \[Data type conversion\]” on page 269](#)

Standards and compatibility

- ◆ **SQL/2003** Transact-SQL extension.

Example

The following statement returns the value 0000009c.

```
SELECT INTTOHEX( 156 );
```

ISDATE function [Data type conversion]

Description

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.

Usage

If a conversion is possible, the function returns 1; otherwise, 0 is returned. If the argument is NULL, 0 is returned.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

ISNULL function [Miscellaneous]

Description

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.

See also

- ◆ [“COALESCE function \[Miscellaneous\]” on page 251](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value -66.

```
SELECT ISNULL( NULL , -66, 55, 45, NULL, 16 );
```

LCASE function [String]

Description

Converts all characters in a string to lowercase. This function is identical the LOWER function.

Syntax

LCASE(*string-expression*)

Parameters

string-expression The string to be converted to lowercase.

See also

- ◆ [“LOWER function \[String\]” on page 280](#)
- ◆ [“UCASE function \[String\]” on page 311](#)
- ◆ [“UPPER function \[String\]” on page 312](#)
- ◆ [“String functions” on page 239](#)

Usage

The LCASE function is similar to the LOWER function.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value chocolate.

```
SELECT LCASE( 'ChoCoLatE' );
```

LEFT function [String]

Description

Returns a number of 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.

Usage

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 column. In this case, the entire value is returned.

Whenever possible, if the input string uses character length semantics the return value will be described in terms of character length semantics.

See also

- ◆ [“RIGHT function \[String\]” on page 295](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the first 5 characters of each Surname value in the Customers table.

```
SELECT LEFT( Surname, 5) FROM Customers;
```

LENGTH function [String]

Description

Returns the number of characters in the specified string.

Syntax

LENGTH(*string-expression*)

Parameters

string-expression The string.

Usage

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 246](#)
- ◆ [“International Languages and Character Sets” \[SQL Anywhere Server - Database Administration\]](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 9.

```
SELECT LENGTH( 'chocolate' );
```

LESSER function [Miscellaneous]

Description

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.

Usage

If the parameters are equal, the first value is returned.

See also

- ◆ [“GREATER function \[Miscellaneous\]” on page 268](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 5.

```
SELECT LESSER( 10,5 ) FROM dummy;
```

LIST function [Aggregate]**Description**

Returns a comma-delimited list of values.

Syntax

```
LIST(  
{ string-expression | DISTINCT string-expression }  
[ , delimiter-string ] )
```

Parameters

string-expression A string, usually a column name. For each row, the expression's value is added to the comma-separated list.

DISTINCT string-expression An expression; for example, the name of a column that you are using in the query. For each unique value of that column, the value is added to the comma-separated list.

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.

Usage

NULL values are not added to the list. LIST (X) 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.

A LIST function cannot be used as a window function, but it can be used as input to a window function.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Examples

The following statement returns all street addresses from the Employees table.

```
SELECT LIST( Street ) FROM Employees;
```

Sorted IDs '1751,591,1062,1191,992,888,318,184,1576,207,1684,1643,1607,1740,409,1507'

LOCATE function [String]

Description

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 $(-1 * \text{offset}) - 1$.

Usage

If *integer-expression* is specified, the search starts at that offset into the string.

The first string can be a long string (longer than 255 bytes), but the second is limited to 255 bytes. If a long string is given as the second argument, the function returns a NULL value. If the string is not found, 0 is returned. Searching for a zero-length string will return 1. If any of the arguments are NULL, the result is NULL.

If multibyte characters are used, with the appropriate collation, then the starting position and the return value may be different from the *byte* positions.

See also

- ◆ [“String functions” on page 239](#)
- ◆ [“CHARINDEX function \[String\]” on page 249](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 8.

```
SELECT LOCATE(  
    'office party this week - rsvp as soon as possible',  
    'party',  
    2 );
```

LOG function [Numeric]

Description

Returns the natural logarithm of a number.

Syntax

LOG(*numeric-expression*)

Parameters

numeric-expression The number.

See also

- ◆ [“LOG10 function \[Numeric\]” on page 279](#)

Usage

The argument is an expression that returns the value of any built-in numeric data type.

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the natural logarithm of 50.

```
SELECT LOG( 50 );
```

LOG10 function [Numeric]

Description

Returns the base 10 logarithm of a number.

Syntax

LOG10(*numeric-expression*)

Parameters

numeric-expression The number.

Usage

The argument is an expression that returns the value of any built-in numeric data type.

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If the parameter is NULL, the result is NULL.

See also

- ◆ [“LOG function \[Numeric\]” on page 278](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the base 10 logarithm for 50.

```
SELECT LOG10( 50 );
```

LOWER function [String]

Description

Converts all characters in a string to lowercase. This function is identical the LCASE function.

Syntax

```
LOWER( string-expression )
```

Parameters

string-expression The string to be converted.

Usage

The LCASE function is identical to the LOWER function.

See also

- ◆ [“LCASE function \[String\]” on page 274](#)
- ◆ [“UCASE function \[String\]” on page 311](#)
- ◆ [“UPPER function \[String\]” on page 312](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature.

Example

The following statement returns the value chocolate.

```
SELECT LOWER( 'chOCOLate' );
```

LTRIM function [String]

Description

Trims leading blanks from a string.

Syntax

```
LTRIM( string-expression )
```

Parameters

string-expression The string to be trimmed.

Usage

The actual length of the result is the length of the expression minus the number of characters removed. If all of 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 297](#)
- ◆ [“TRIM function \[String\]” on page 310](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

The TRIM specifications defined by the SQL/2003 standard (LEADING and TRAILING) are supplied by the SQL Anywhere LTRIM and RTRIM functions respectively.

Example

The following statement returns the value Test Message with all leading blanks removed.

```
SELECT LTRIM( '   Test Message' );
```

MAX function [Aggregate]

Description

Returns the maximum *expression* value found in each group of rows.

Syntax 1

```
MAX( expression | 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.

Usage

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

See also

- ◆ [“MIN function \[Aggregate\]” on page 282](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature. Syntax 2 is feature T611.

Example

The following statement returns the value 138948.000, representing the maximum salary in the Employees table.

```
SELECT MAX( Salary )
FROM Employees;
```

MIN function [Aggregate]

Description

Returns the minimum expression value found in each group of rows.

Syntax 1

```
MIN( expression | 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.

Usage

Rows where *expression* is NULL are ignored. Returns NULL for a group containing no rows.

See also

- ◆ [“MAX function \[Aggregate\]” on page 281](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature. Syntax 2 is feature T611.

Example

The following statement returns the value 24903.000, representing the minimum salary in the Employees table.

```
SELECT MIN( Salary )
FROM Employees;
```

MINUTE function [Date and time]

Description

Returns a minute component of a datetime value.

Syntax

```
MINUTE( datetime-expression )
```

Parameters

datetime-expression The datetime value.

Usage

The value returned is a number from number from 0 to 59 corresponding to the datetime minute.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 22.

```
SELECT MINUTE( '1998-07-13 12:22:34' );
```

MINUTES function [Date and time]**Description**

The behavior of this function can vary depending on what you supply:

- ◆ If you give a single date, this function returns the number of minutes since 0000-02-29.

Note

0000-02-29 is not meant to imply an actual date; it is the date used by the date algorithm.

- ◆ If you give two timestamps, this function returns the integer number of minutes between them. Instead, use the DATEDIFF function.
- ◆ If you give a date and an integer, this function adds the integer number of minutes to the specified timestamp. Instead, use the DATEADD function.

Syntax 1: integer

```
MINUTES( [ datetime-expression, ] datetime-expression )
```

Syntax 2: timestamp

```
MINUTES( datetime-expression, integer-expression )
```

Parameters

datetime-expression A date and time.

integer-expression The number of minutes to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of minutes is subtracted from the datetime value. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a DATETIME data type.

Usage

Since this function returns an integer, overflow can occur when syntax 1 is used with timestamps greater than or equal to 4083-03-23 02:08:00.

See also

- ◆ [“CAST function \[Data type conversion\]” on page 247](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statements return the value 240, signifying that the second timestamp is 240 seconds 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 1999-05-12 21:10:07.000. It is recommended that you use the second example (DATEADD).

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07'  
                    AS DATETIME ), 5);
```

```
SELECT DATEADD( minute, 5, '1999-05-12 21:05:07' );
```

MOD function [Numeric]

Description

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.

Usage

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 292](#)

Standards and compatibility

- ◆ **SQL/2003** SQL foundation feature outside of core SQL.

Example

The following statement returns the value 2.

```
SELECT MOD( 5, 3 );
```

MONTH function [Date and time]**Description**

Returns a month of the given date.

Syntax

```
MONTH( date-expression )
```

Parameters

date-expression A datetime value.

Usage

The value returned is a number from number from 1 to 12 corresponding to the datetime month.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 7.

```
SELECT MONTH( '1998-07-13' );
```

MONTHNAME function [Date and time]**Description**

Returns the name of the month from a date.

Syntax

```
MONTHNAME( date-expression )
```

Parameters

date-expression The datetime value.

Usage

The MONTHNAME function returns a string, even if the result is numeric, such as 2 for the month of February.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value September.

```
SELECT MONTHNAME( '1998-09-05' );
```

MONTHS function [Date and time]

Description

The behavior of this function can vary depending on what you supply:

- ◆ If you give a single date, this function returns the number of months since 0000-02.

Note

0000-02 is not meant to imply an actual date; it is the date used by the date algorithm.

- ◆ If you give two timestamps, this function returns the integer number of months between them. Instead, use the DATEDIFF function.
- ◆ If you give a date and an integer, this function adds the integer number of minutes to the specified timestamp. Instead, use the DATEADD function.

Syntax 1: integer

```
MONTHS( [ datetime-expression, ] datetime-expression )
```

Syntax 2: timestamp

```
MONTHS( datetime-expression, integer-expression )
```

Parameters

datetime-expression A date and time.

integer-expression The number of months to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of months is subtracted from the datetime value. If you supply an *integer-expression*, the *datetime-expression* must be explicitly cast as a datetime data type.

☞ For information on casting data types, see [“CAST function \[Data type conversion\]” on page 247](#).

Usage

The value of MONTHS is calculated from the number of first days of the month between the two dates.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statements return the value 2, signifying that the second date is two months after the first. It is recommended that you use the second example (DATEDIFF).

```
SELECT MONTHS( '1999-07-13 06:07:12',  
              '1999-09-13 10:07:12' );
```

```
SELECT DATEDIFF( month,  
              '1999-07-13 06:07:12',  
              '1999-09-13 10:07:12' );
```

The following statement returns the value 23981.

```
SELECT MONTHS( '1998-07-13 06:07:12' );
```

The following statements return the timestamp 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]

Description

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.

Usage

The NEWID function generates a unique identifier value. It can be used in a DEFAULT clause for a column.

UUIDs can be used to uniquely identify rows in a table. The values are generated such that a value produced on one computer will not match that produced on another. Hence, they can also be used as keys in synchronization and replication environments.

See also

- ◆ [“The NEWID default” \[SQL Anywhere Server - SQL Usage\]](#)
- ◆ [“STRTOUUID function \[String\]” on page 305](#)
- ◆ [“UIDTOSTR function \[String\]” on page 313](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement creates a table named mytab with two columns. Column pk has a unique identifier data type, and assigns the NEWID function as the default value. Column c1 has an integer data type.

```
CREATE TABLE mytab(  
    pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),  
    c1 INT );
```

The following statement returns a unique identifier as a string:

```
SELECT NEWID();
```

For example, the value returned might be 96603324-6FF6-49DE-BF7D-F44C1C7E6856.

NOW function [Date and time]

Description

Returns the current year, month, day, hour, minute, second, and fraction of a second. The accuracy is limited by the accuracy of the system clock.

Syntax

```
NOW( * )
```

Usage

The information the NOW function returns is equivalent to the information returned by the GETDATE function and the CURRENT_TIMESTAMP special value.

See also

- ◆ [“GETDATE function \[Date and time\]” on page 267](#)
- ◆ [“CURRENT_TIMESTAMP special value” on page 209](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the current date and time.

```
SELECT NOW( * );
```

NULLIF function [Miscellaneous]

Description

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.

Usage

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 217](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature.

Example

The following statement returns the value a:

```
SELECT NULLIF( 'a', 'b' );
```

The following statement returns NULL.

```
SELECT NULLIF( 'a', 'a' );
```

PATINDEX function [String]**Description**

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 not.

The pattern for UltraLite uses the following wildcards:

Wildcard	Matches
_ (underscore)	Any one character
% (percent)	Any string of zero or more characters

Wildcard	Matches
[]	Any single character in the specified range or set
[^]	Any single character not in the specified range or set

string-expression The string to be searched for the pattern.

Usage

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 278](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 2.

```
SELECT PATINDEX( '%hoco%', 'chocolate' );
```

The following statement returns the value 11.

```
SELECT PATINDEX( '%4_5_', '0a1A 2a3A 4a5A' );
```

PI function [Numeric]

Description

Returns the numeric value PI.

Syntax

```
PI( * )
```

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Usage

This function returns a DOUBLE value.

Example

The following statement returns the value 3.141592653...

```
SELECT PI( * );
```

POWER function [Numeric]

Description

Calculates one number raised to the power of another.

Syntax

POWER(*numeric-expression-1*, *numeric-expression-2*)

Parameters

numeric-expression-1 The base.

numeric-expression-2 The exponent.

Usage

This function converts its arguments to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result. If any argument is NULL, the result is a NULL value.

Standards and compatibility

◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 64.

```
SELECT POWER( 2, 6 );
```

QUARTER function [Date and time]

Description

Returns a number indicating the quarter of the year from the supplied date expression.

Syntax

QUARTER(*date-expression*)

Parameters

date-expression The date.

Usage

The quarters are as follows:

Quarter	Period (inclusive)
1	January 1 to March 31
2	April 1 to June 30
3	July 1 to September 30

Quarter	Period (inclusive)
4	October 1 to December 31

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 2.

```
SELECT QUARTER( '1987/05/02' );
```

RADIANS function [Numeric]

Description

Converts a number from degrees to radians.

Syntax

```
RADIANS( numeric-expression )
```

Parameters

numeric-expression A number, in degrees. This angle is converted to radians.

Usage

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns a value of approximately 0.5236.

```
SELECT RADIANS( 30 );
```

REMAINDER function [Numeric]

Description

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.

Usage

Alternatively, try using the MOD function.

See also

- ◆ [“MOD function \[Numeric\]” on page 284](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 2.

```
SELECT REMAINDER( 5, 3 );
```

REPEAT function [String]

Description

Concatenates a string a specified number of times.

Syntax

```
REPEAT( string-expression, integer-expression )
```

Parameters

string-expression The string to be repeated.

integer-expression The number of times the string is to be repeated. If *integer-expression* is negative, an empty string is returned.

Usage

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.

Alternatively, try using the REPLICATE function.

See also

- ◆ [“REPLICATE function \[String\]” on page 295](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value repeatrepeatrepeat.

```
SELECT REPEAT( 'repeat', 3 );
```

REPLACE function [String]

Description

Replaces a string with another string, and returns the new results.

Syntax

```
REPLACE( original-string, search-string, replace-string )
```

Parameters

If any argument is NULL, the function returns NULL.

original-string The string to be searched. This can be any length.

search-string The string to be searched for and replaced with *replace-string*. This string is limited to 255 bytes. If *search-string* is an empty string, the original string is returned unchanged.

replace-string The replacement string, which replaces *search-string*. This can be any length. If *replacement-string* is an empty string, all occurrences of *search-string* are deleted.

Usage

This function replaces all occurrences.

See also

- ◆ [“SUBSTRING function \[String\]” on page 307](#)
- ◆ [“CHARINDEX function \[String\]” on page 249](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value xx.def.xx.ghi.

```
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.)

```
SELECT REPLACE(
    REPLACE( proc_defn, 'OldTableName', 'NewTableName' ),
    'CREATE PROCEDURE',
    'ALTER PROCEDURE' )
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%';
```

Use a separator other than the comma for the LIST function:

```
SELECT REPLACE( LIST( table_id ), ',', '---')
FROM SYS.SYSTAB
WHERE table_id <= 5;
```

REPLICATE function [String]

Description

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.

Usage

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.

Alternatively, try using the REPEAT function.

See also

- ◆ [“REPEAT function \[String\]” on page 293](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value repeatrepeatrepeat.

```
SELECT REPLICATE( 'repeat', 3 );
```

RIGHT function [String]

Description

Returns the rightmost characters of a string.

Syntax

```
RIGHT( string-expression, integer-expression )
```

Parameters

string-expression The string to be left-truncated.

integer-expression The number of characters at the end of the string to return.

Usage

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 column. In this case, the entire value is returned.

Whenever possible, if the input string uses character length semantics the return value will be described in terms of character length semantics.

See also

- ◆ [“LEFT function \[String\]” on page 275](#)
- ◆ [“International Languages and Character Sets” \[SQL Anywhere Server - Database Administration\]](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the last 5 characters of each Surname value in the Customers table.

```
SELECT RIGHT( Surname, 5) FROM Customers;
```

ROUND function [Numeric]

Description

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.

Usage

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 311](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 123.200.

```
SELECT ROUND( 123.234, 1 );
```

RTRIM function [String]**Description**

Returns a string with trailing blanks removed.

Syntax

```
RTRIM( string-expression )
```

Parameters

string-expression The string to be trimmed.

Usage

The actual length of the result is the length of the expression minus the number of characters removed. If all of 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 310](#)
- ◆ [“LTRIM function \[String\]” on page 280](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

The TRIM specifications defined by the SQL/2003 standard (LEADING and TRAILING) are supplied by the SQL Anywhere LTRIM and RTRIM functions respectively.

Example

The following statement returns the string Test Message, with all trailing blanks removed.

```
SELECT RTRIM( 'Test Message      ' );
```

SECOND function [Date and time]**Description**

Returns a second of the given date.

Syntax**SECOND**(*datetime-expression*)**Parameters****datetime-expression** The datetime value.**Usage**

Returns a number from 0 to 59 corresponding to the second component of the given datetime value.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 25.

```
SELECT SECOND( '1998-07-13 21:21:25' );
```

SECONDS function [Date and time]

Description

The behavior of this function can vary depending on what you supply:

- ◆ If you give a single date, this function returns the number of seconds since 0000-02-29.

Note

0000-02-29 is not meant to imply an actual date; it is the date used by the date algorithm.

- ◆ If you give two timestamps, this function returns the integer number of seconds between them. Instead, use the DATEDIFF function.
- ◆ If you give a date and an integer, this function adds the integer number of seconds to the specified timestamp. Instead, use the DATEADD function.

Syntax 1: integer**SECONDS**([*datetime-expression*,] *datetime-expression*)**Syntax 2: timestamp****SECONDS**(*datetime-expression*, *integer-expression*)**Parameters****datetime-expression** A date and time.**integer-expression** The number of seconds to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of minutes is subtracted from the datetime value. If you supply an integer expression, the *datetime-expression* must be explicitly cast as a datetime data type.

See also

- ◆ [“CAST function \[Data type conversion\]” on page 247](#)
- ◆ [“DATEADD function \[Date and time\]” on page 257](#)
- ◆ [“DATEDIFF function \[Date and time\]” on page 258](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statements return the value 14400, signifying that the second timestamp is 14400 seconds after the first.

```
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.

```
SELECT SECONDS( '1998-07-13 06:07:12' );
```

The following statements return the datetime 1999-05-12 21:05:12.0.

```
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]

Description

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 )
```

Usage

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.

See also

- ◆ [“EXPLANATION function \[Miscellaneous\]” on page 266](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement passes a **SELECT** statement as a string parameter and returns the plan for executing the query.

```
SELECT EXPLANATION(  
    'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

This information can help with decisions about indexes to add or how to structure your database for better performance.

In Interactive SQL, you can view the plan for any SQL statement on the Plan tab in the Results pane.

SIGN function [Numeric]

Description

Returns the sign of a number.

Syntax

```
SIGN( numeric-expression )
```

Parameters

numeric-expression The number for which the sign is to be returned.

Usage

For negative numbers, the **SIGN** function returns -1.

For zero, the **SIGN** function returns 0.

For positive numbers, the **SIGN** function returns 1.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value -1

```
SELECT SIGN( -550 );
```

SIMILAR function [String]

Description

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.

Usage

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. Join the table to itself and produce a report of all similarities greater than 90 percent, but less than 100 percent.

The calculation performed for the SIMILAR function is more complex than just the number of characters that match.

See also

- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 75, indicating that the two values are 75% similar.

```
SELECT SIMILAR( 'toast', 'coast' );
```

SIN function [Numeric]

Description

Returns the sine of a number.

Syntax

SIN(*numeric-expression*)

Parameters

numeric-expression The angle, in radians.

Usage

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, and returns a DOUBLE as the result.

See also

- ◆ [“ASIN function \[Numeric\]” on page 243](#)
- ◆ [“COS function \[Numeric\]” on page 254](#)
- ◆ [“COT function \[Numeric\]” on page 255](#)
- ◆ [“TAN function \[Numeric\]” on page 309](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the SIN value of 0.52.

```
SELECT SIN( 0.52 );
```

SOUNDEX function [String]

Description

Returns a number representing the sound of a string.

Syntax

SOUNDEX(*string-expression*)

Parameters

string-expression The string.

Usage

The SOUNDEX function value for a string is based on the first letter and the next three consonants other than H, Y, and W. Vowels in *string-expression* are ignored unless they are the first letter of the string. Doubled letters are counted as one letter. For example, the word apples is based on the letters A, P, L, and S.

Multibyte characters are ignored by the SOUNDEX function.

Although it is not perfect, the SOUNDEX function normally returns the same number for words that sound similar and that start with the same letter.

The SOUNDEX function works best with English words. It is less useful for other languages.

See also

- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns two identical numbers, 3827, representing the sound of each name.

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' );
```

SPACE function [String]**Description**

Returns a specified number of spaces.

Syntax

```
SPACE( integer-expression )
```

Parameters

integer-expression The number of spaces to return.

Usage

If *integer-expression* is negative, a null string is returned.

See also

- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns a string containing 10 spaces.

```
SELECT SPACE( 10 );
```

SQRT function [Numeric]**Description**

Returns the square root of a number.

Syntax

```
SQRT( numeric-expression )
```

Parameters

numeric-expression The number for which the square root is to be calculated.

Usage

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 3.

```
SELECT SQRT( 9 );
```

STR function [String]

Description

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.

Usage

If the integer portion of the number cannot fit in the length specified, then the result is a string of the specified length containing all asterisks. For example, the following statement returns ***.

```
SELECT STR( 1234.56, 3 );
```

Note

The maximum length that is supported is 128. Any length that is not between 1 and 128 yields a result of NULL.

See also

- ◆ [“String functions” on page 239](#)

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 );
```


Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

STRING function [String]**Description**

Concatenates one or more strings into one large string.

Syntax

STRING(*string-expression*,...)

Parameters

string-expression A string.

If only one argument is supplied, it is converted into a single expression. If more than one argument is supplied, they are concatenated into a single string.

Usage

Numeric or date parameters are converted to strings before concatenation. The STRING function can also be used to convert any single expression to a string by supplying that expression as the only parameter.

If all parameters are NULL, STRING returns NULL. If any parameters are non-NULL, then any NULL parameters are treated as empty strings.

See also

- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value *testing123*.

```
SELECT STRING( 'testing', NULL, 123 );
```

STRTOUUID function [String]**Description**

Converts a string value to a unique identifier (UUID or GUID) value.

Not needed in newer databases

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 [“Data types in UltraLite” on page 212](#).

Syntax

STRTOUUID(*string-expression*)

Parameters

string-expression A string in the format `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Usage

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 313](#)
- ◆ [“NEWID function \[Miscellaneous\]” on page 287](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

```
CREATE TABLE T1 (  
    pk UNIQUEIDENTIFIER PRIMARY KEY, c1 INT );  
INSERT INTO T1 ( pk, c1 )  
VALUES ( STRTOUUID('12345678-1234-5678-9012-123456789012'), 1 );
```

STUFF function [String]

Description

Deletes a number of 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.

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.

Usage

See also

- ◆ [“INSERTSTR function \[String\]” on page 272](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value chocolate pie.

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' );
```

SUBSTRING function [String]

Description

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.

Usage

In UltraLite, the database does not have an `ansi_substring` option. Nonetheless, the SUBSTR function behaves as if `ansi_substring` is set to on by default. In other words, the function's behavior corresponds to ANSI/ISO SQL/2003 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 *length* returns an error.

If *string-expression* is of binary data type, the SUBSTRING function behaves as BYTE_SUBSTR.

It is recommended that you avoid using non-positive start offsets or negative lengths with the SUBSTRING function. Where possible, use the LEFT or RIGHT functions instead.

Whenever possible, if the input string uses character length semantics the return value is described in terms of character length semantics.

See also

- ◆ [“BYTE_SUBSTR function \[String\]” on page 246](#)
- ◆ [“LEFT function \[String\]” on page 275](#)
- ◆ [“RIGHT function \[String\]” on page 295](#)
- ◆ [“CHARINDEX function \[String\]” on page 249](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature.

Example

The following table shows the values returned by the SUBSTRING function when used in a SELECT statement, with the ansi_substring option set to On and Off.

Example	Result with ansi_substring set to On	Result with ansi_substring set to Off
SUBSTRING('front yard', 1, 4)	fron	fron
SUBSTRING('back yard', 6, 4)	yard	yard
SUBSTR('abcdefgh', 0, -2)	Returns an error	gh
SUBSTR('abcdefgh', -2, 2)	Returns an empty string	gh
SUBSTR('abcdefgh', 2, -2)	Returns an error	ab
SUBSTR('abcdefgh', 2, -4)	Returns an error	ab
SUBSTR('abcdefgh', 2, -1)	Returns an error	b

SUM function [Aggregate]

Description

Returns the total of the specified expression for each group of rows.

Syntax 1

SUM(*expression* | **DISTINCT** *expression*)

Parameters

expression The object to be summed. This is commonly a column name.

DISTINCT expression Computes the sum of the unique values of *expression* in the input.

Usage

Rows where the specified expression is NULL are not included.

Returns NULL for a group containing no rows.

See also

- ◆ [“COUNT function \[Aggregate\]” on page 255](#)
- ◆ [“AVG function \[Aggregate\]” on page 245](#)

Standards and compatibility

- ◆ **SQL/2003** Core feature. Syntax 2 is T611.

Example

The following statement returns the value 3749146.740.

```
SELECT SUM( Salary )  
FROM Employees;
```

TAN function [Numeric]

Description

Returns the tangent of a number.

Syntax

TAN(*numeric-expression*)

Parameters

numeric-expression An angle, in radians.

Usage

The ATAN and TAN functions are inverse operations.

This function converts its argument to DOUBLE, performs the computation in double-precision floating point, and returns a DOUBLE as the result.

See also

- ◆ [“COS function \[Numeric\]” on page 254](#)
- ◆ [“SIN function \[Numeric\]” on page 301](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value of the tan of 0.52.

```
SELECT TAN( 0.52 );
```

TODAY function [Date and time]

Description

Returns the current date.

Syntax

```
TODAY( * )
```

Usage

Use this syntax in place of the historical CURRENT DATE function.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statements return the current day according to the system clock.

```
SELECT TODAY( * ) ;  
SELECT CURRENT DATE
```

TRIM function [String]

Description

Removes leading and trailing blanks from a string.

Syntax

```
TRIM( string-expression )
```

Parameters

string-expression The string to be trimmed.

Usage

See also

- ◆ [“LTRIM function \[String\]” on page 280](#)
- ◆ [“RTRIM function \[String\]” on page 297](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** The TRIM function is a SQL/2003 core feature.

SQL Anywhere does not support the additional parameters *trim specification* and *trim character*, as defined in SQL/2003. The SQL Anywhere implementation of TRIM corresponds to a TRIM specification of BOTH.

For the other TRIM specifications defined by the SQL/2003 standard (LEADING and TRAILING), SQL Anywhere supplies the LTRIM and RTRIM functions respectively.

Example

The following statement returns the value chocolate with no leading or trailing blanks.

```
SELECT TRIM( '  chocolate  ' );
```

TRUNCNUM function [Numeric]

Description

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.

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.

Usage

If any parameter is NULL, the result is NULL.

See also

◆ [“ROUND function \[Numeric\]” on page 296](#)

Standards and compatibility

◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value 600.

```
SELECT TRUNCNUM( 655, -2 );
```

The following statement: returns the value 655.340.

```
SELECT TRUNCNUM( 655.348, 2 );
```

UCASE function [String]

Description

Converts all characters in a string to uppercase. This function is identical the UPPER function.

Syntax

UCASE(*string-expression*)

Parameters

string-expression The string to be converted to uppercase.

Usage

The UCASE function is similar to the UPPER function.

See also

- ◆ [“UPPER function \[String\]” on page 312](#)
- ◆ [“LCASE function \[String\]” on page 274](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value CHOCOLATE.

```
SELECT UCASE( 'ChocoLate' );
```

UPPER function [String]

Description

Converts all characters in a string to uppercase. This function is identical the UCASE function.

Syntax

UPPER(*string-expression*)

Parameters

string-expression The string to be converted to uppercase.

Usage

The UCASE function is similar to the UPPER function.

See also

- ◆ [“UCASE function \[String\]” on page 311](#)
- ◆ [“LCASE function \[String\]” on page 274](#)
- ◆ [“LOWER function \[String\]” on page 280](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement returns the value CHOCOLATE.

```
SELECT UPPER( 'ChocoLate' );
```

UUIDTOSTR function [String]**Description**

Converts a unique identifier value (UUID, also known as GUID) to a string value.

Not needed in newer databases

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 [“Data types in UltraLite” on page 212](#).

Syntax

```
UUIDTOSTR( uuid-expression )
```

Parameters

uuid-expression A unique identifier value.

Usage

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 if you want to view a UUID value.

See also

- ◆ [“NEWID function \[Miscellaneous\]” on page 287](#)
- ◆ [“STRTOUUID function \[String\]” on page 305](#)
- ◆ [“String functions” on page 239](#)

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statement creates a table mytab with two columns. Column pk has a unique identifier data type, and column c1 has an integer data type. It then inserts two rows with the values 1 and 2 respectively into column c1.

```
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 of the data in the newly created table.

```
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 command:

```
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]

Description

Given two dates, this function returns the integer number of weeks between them. It is recommended that you use the [“DATEDIFF function \[Date and time\]” on page 258](#) instead for this purpose.

Given a single date, this function returns the number of weeks since 0000-02-29.

Given one date and an integer, it adds the integer number of weeks to the specified date. It is recommended that you use the [“DATEADD function \[Date and time\]” on page 257](#) instead for this purpose.

Syntax 1 returns an integer. Syntax 2 returns a timestamp.

Syntax 1

```
WEEKS( [ datetime-expression, ] datetime-expression )
```

Syntax 2

```
WEEKS( datetime-expression, integer-expression )
```

Parameters

datetime-expression A date and time.

integer-expression The number of weeks to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of weeks is subtracted from the datetime value. If you supply an *integer-expression*, the *datetime-expression* must be explicitly cast as a datetime data type.

 For information on casting data types, see [“CAST function \[Data type conversion\]” on page 247](#).

Usage

The difference of two dates in weeks is the number of Sundays between the two dates.

Standards and compatibility

◆ **SQL/2003** Vendor extension.

Example

The following statements return the value 8, signifying that the second date is eight weeks after the first. It is recommended that you use the second form (DATEDIFF).

```
SELECT WEEKS( '1999-07-13 06:07:12',  
             '1999-09-13 10:07:12' );
```

```
SELECT DATEDIFF( week,  
               '1999-07-13 06:07:12',  
               '1999-09-13 10:07:12' );
```

The following statement returns the value 104270.

```
SELECT WEEKS( '1998-07-13 06:07:12' );
```

The following statements return the timestamp 1999-06-16 21:05:07.0. It is recommended that you use the second form (DATEADD).

```
SELECT WEEKS( CAST( '1999-05-12 21:05:07'  
                  AS TIMESTAMP ), 5);
```

```
SELECT DATEADD( week, 5, '1999-05-12 21:05:07' );
```

YEAR function [Date and time]

Description

Takes a timestamp value as a parameter and returns the year specified by that timestamp.

Syntax

```
YEAR( datetime-expression )
```

Parameters

datetime-expression A date, time, or timestamp.

Usage

The value is returned as a short value.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following example returns the value 2001.

```
SELECT YEAR( '2001-09-12' );
```

YEARS function [Date and time]

Description

Given two dates, this function returns the integer number of years between them. It is recommended that you use the [“DATEDIFF function \[Date and time\]” on page 258](#) instead for this purpose.

Given one date, it returns the year. It is recommended that you use the [“DATEPART function \[Date and time\]” on page 260](#) instead for this purpose.

Given one date and an integer, it adds the integer number of years to the specified date. It is recommended that you use the [“DATEADD function \[Date and time\]” on page 257](#) instead for this purpose.

Syntax 1

YEARS([*datetime-expression*,] *datetime-expression*)

Syntax 2

YEARS(*datetime-expression*, *integer-expression*)

Parameters

datetime-expression A date and time.

integer-expression The number of years to be added to the *datetime-expression*. If *integer-expression* is negative, the appropriate number of years is subtracted from the datetime value. If you supply an *integer-expression*, the *datetime-expression* must be explicitly cast as a datetime data type.

 For information on casting data types, see [“CAST function \[Data type conversion\]” on page 247](#).

Usage

The value of YEARS is calculated from the number of first days of the year between the two dates.

Syntax 1 returns an integer. Syntax 2 returns a timestamp.

Standards and compatibility

- ◆ **SQL/2003** Vendor extension.

Example

The following statements both return -4.

```
SELECT YEARS( '1998-07-13 06:07:12',
              '1994-03-13 08:07:13' );

SELECT DATEDIFF( year,
                 '1998-07-13 06:07:12',
                 '1994-03-13 08:07:13' );
```

The following statements return 1998.

```
SELECT YEARS( '1998-07-13 06:07:12' )
SELECT DATEPART( year, '1998-07-13 06:07:12' );
```

The following statements return the given date advanced 300 years.

```
SELECT YEARS( CAST( '1998-07-13 06:07:12' AS TIMESTAMP ), 300 )
SELECT DATEADD( year, 300, '1998-07-13 06:07:12' );
```

YMD function [Date and time]

Description

Returns a date value corresponding to the given year, month, and day of the month. Values are small integers from -32768 to 32767.

Syntax

```
YMD(  
integer-expression1,  
integer-expression2,  
integer-expression3 )
```

Parameters

integer-expression1 The year.

integer-expression2 The number of the month. If the month is outside the range 1–12, the year is adjusted accordingly.

integer-expression3 The day number. The day can be any integer; the date is adjusted accordingly.

Standards and compatibility

◆ **SQL/2003** Vendor extension.

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 will adjust accordingly. For example, the following statement returns the value 2000-03-01.

```
SELECT YMD( 1999, 15, 1 );
```

CHAPTER 12

UltraLite SQL Statement Reference

Contents

UltraLite SQL statements overview	320
ALTER TABLE statement	322
ALTER PUBLICATION statement	326
COMMIT statement	327
CREATE INDEX statement	328
CREATE PUBLICATION statement	330
CREATE TABLE statement	331
DELETE statement	336
DROP INDEX statement	337
DROP PUBLICATION statement	338
DROP TABLE statement	339
INSERT statement	340
ROLLBACK statement	341
SELECT statement	342
START SYNCHRONIZATION DELETE statement	347
STOP SYNCHRONIZATION DELETE statement	348
TRUNCATE TABLE statement	349
UNION operation	351
UPDATE statement	352

About this chapter

This chapter provides a complete reference for the SQL language used by UltraLite. Specifically, it describes the syntax and conventions required in an UltraLite context.

UltraLite SQL statements overview

The SQL statements in this chapter are supported by UltraLite SQL. They are a subset of the statement 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.
- ◆ UltraLite SQL statements follow the same syntax conventions used by SQL Anywhere statements. Ensure you understand these conventions and how they are used to represent SQL syntax. See [“Syntax conventions” \[SQL Anywhere Server - SQL Reference\]](#).
- ◆ Using UltraLite SQL creates a transaction. A transaction consists of all changes (INSERTs, UPDATEs, and DELETEs) since the last ROLLBACK or COMMIT. For more comprehensive details, see [“Transaction processing, recovery, and backup” on page 19](#).

These changes can be made permanent by executing a COMMIT. A ROLLBACK statement causes the changes to be removed. See [“COMMIT statement” on page 327](#) and [“ROLLBACK statement” on page 341](#).

Statement categories

SQL statements are organized and identified by the initial word in a statement, which is almost always a verb. This 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” on page 342](#).
 For more information on supported UltraLite SQL expressions, see [“Expressions in UltraLite” on page 215](#). For more information on supported UltraLite SQL operators see [“Operators in UltraLite” on page 228](#).
- ◆ **Data manipulation statements** Allow you to change content in the database. Data manipulation is achieved with the [“DELETE statement” on page 336](#), the [“INSERT statement” on page 340](#), and the [“UPDATE statement” on page 352](#).
- ◆ **Data definition statements** Allow you to define the structure or schema of a database. The schema can be changed with the [“CREATE INDEX statement” on page 328](#), the [“CREATE TABLE statement” on page 331](#), the [“DROP INDEX statement” on page 337](#) the [“DROP TABLE statement” on page 339](#), and even the [“ALTER TABLE statement” on page 322](#) and the [“TRUNCATE TABLE statement” on page 349](#).
- ◆ **Transaction control statements** Allow you to control transactions within your UltraLite application. Transaction control is achieved with either the [“COMMIT statement” on page 327](#) or the [“ROLLBACK statement” on page 341](#).

- ◆ **Synchronization management** Allow you to temporarily control synchronization with a MobiLink server. Synchronization management is achieved with the [“START SYNCHRONIZATION DELETE statement” on page 347](#), [“STOP SYNCHRONIZATION DELETE statement” on page 348](#), the [“ALTER PUBLICATION statement” on page 326](#), the [“CREATE PUBLICATION statement” on page 330](#), and the [“DROP PUBLICATION statement” on page 338](#).

ALTER TABLE statement

Description

Use this statement to modify 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*

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*]
[*column-constraint* ...]

column-constraint :
[**UNIQUE**]

column-default :
{ **GLOBAL AUTOINCREMENT** [(*number*)] |
AUTOINCREMENT | **CURRENT DATE** |
CURRENT TIME | **CURRENT TIMESTAMP** |
NULL |
NEWID() |
constant-value }

table-constraint :
[**CONSTRAINT** *constraint-name*]
{ *foreign-key-constraint* | *unique-key-constraint* }
[**WITH MAX HASH SIZE** *value*]

foreign-key-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**], ...,)]

Parameters

add-clause Add a new column or table constraint to the table:

- ◆ **ADD *column-definition*** 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.
- ◆ **ADD *table-constraint*** 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.

Note

You cannot add a primary key in UltraLite.

☞ See “[CREATE TABLE statement](#)” on page 331 for a full explanation of *column-definition* and *table-constraint*.

modify-clause Change a single column definition.

☞ See “[CREATE TABLE statement](#)” on page 331 for a full explanation of *column-definition* (note that primary keys in the *column-definition* cannot be used with ALTER statements).

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 index, constraint, or key must be deleted before the column can be deleted.
- ◆ **DROP CONSTRAINT *table-constraint*** Delete the named constraint from the table definition.

Note

You cannot drop a primary key in UltraLite.

☞ See “[CREATE TABLE statement](#)” on page 331 for a full explanation of *table-constraint*.

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*. Note that 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*. Note that 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*. Note that 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 in order to help 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.

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.

If the column is contained in a uniqueness constraint, a foreign key, or a primary key, then the constraint or key must be deleted before the column can be modified. You cannot MODIFY a table or column constraint. To change a constraint, you must DELETE the old constraint and ADD the new constraint.

A table whose name ends with **nosync** can only be renamed to a table name that also ends with **nosync**. See [“Nosync tables in UltraLite” \[MobiLink - Client Administration\]](#).

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 [“Dispose method” \[UltraLite - .NET Programming\]](#).

Statements are not released if schema changes are initiated at the same time. See [“Impact of schema changes in UltraLite” on page 15](#).

Examples

The following example drops the office column from the employee table.

```
ALTER TABLE employee
DROP office
```

The following example allows the table to now hold up to 50 characters.

```
ALTER TABLE customer
MODIFY address CHAR(50)
```

See also

- ◆ [“CREATE TABLE statement” on page 331](#)
- ◆ [“DROP TABLE statement” on page 339](#)
- ◆ [“Data types in UltraLite” on page 212](#)
- ◆ [“Altering UltraLite column definitions” on page 59](#)

- ◆ “Using table and column constraints” [*SQL Anywhere Server - SQL Usage*]
- ◆ “Overriding partition sizes for autoincremented columns” [*MobiLink - Client Administration*]
- ◆ “Determining the most recently assigned GLOBAL AUTOINCREMENT value” [*MobiLink - Client Administration*]

ALTER PUBLICATION statement

Description

Use this statement to alter a publication. A publication identifies synchronized data in an UltraLite remote database.

Syntax

```
ALTER PUBLICATION publication-name alterpub-clause
```

alterpub-clause :

```
  ADD TABLE table-name [ WHERE search-condition ]  
| ALTER TABLE table-name [ WHERE search-condition ]  
| { DELETE | DROP } table-name  
| RENAME publication-name
```

Parameters

WHERE clause If a WHERE clause is changed via this statement, only rows satisfying *search-condition* are considered for upload from the associated table during synchronization. For information about search conditions, see [“Search conditions in UltraLite” on page 221](#).

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.

Side effects

Automatic commit.

See also

- ◆ [“Designing synchronization in UltraLite” \[MobiLink - Client Administration\]](#)
- ◆ [“CREATE PUBLICATION statement” on page 330](#)
- ◆ [“DROP PUBLICATION statement” on page 338](#)
- ◆ [“START SYNCHRONIZATION DELETE statement” on page 347](#)
- ◆ [“STOP SYNCHRONIZATION DELETE statement” on page 348](#)

Example

The following statement adds the customers table to the pub_contact publication.

```
ALTER PUBLICATION pub_contact  
  ADD TABLE customers
```

COMMIT statement

Description

Use this statement to make changes to the database permanent.

Syntax

```
COMMIT [ WORK ]
```

Parameters

WORK is an optional keyword.

Remarks

A transaction is the inserts, updates, and deletes performed on one database connection to a database between COMMIT or ROLLBACK statements. The COMMIT statement ends a transaction and makes all changes made during this transaction permanent in the database.

ALTER, CREATE, and DROP statements are committed automatically.


See also

- ◆ [“ROLLBACK statement” on page 341](#)

CREATE INDEX statement

Description

Use this statement to create an index on a specified table. 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 INSERT, DELETE, and UPDATE statements, as well as synchronization.

 For more information about indexes, see [“Indexes in UltraLite databases” on page 20](#). UltraLite can also use query access plans to optimize queries. For details, see [“Query access plans in UltraLite” on page 232](#).

Syntax

```
CREATE [ UNIQUE ] INDEX [ index-name ]  
ON table-name ( ordered-column-list )  
WITH MAX HASH SIZE x
```

ordered-column-list :
[(*column-name* [**ASC** | **DESC**], ...,)]

Parameters

UNIQUE The UNIQUE parameter 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.

There is a difference between a unique constraint on a table and a unique index. Columns in a unique index are allowed to be NULL, while columns in a unique constraint are not. A foreign key can reference either a primary key or a column with a unique constraint, but not a unique index, because it can include multiple instances of NULL.

ordered-column-list An ordered list of columns. The ordered list can be sorted in ascending or descending order.

WITH MAX HASH SIZE Sets the default index hash size in bytes. If you do not set this value, a default size of 8 bytes is used for index hashing. See [“max_hash_size property” on page 106](#).

Remarks

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.

For UltraLite.NET users: You cannot execute this statement, unless you also call the Dispose method for all data objects (for example, ULDataReader). See [“Dispose method” \[UltraLite - .NET Programming\]](#).

UltraLite automatically creates indexes for primary keys and for unique constraints.

Statements are not released if schema changes are initiated at the same time. See [“Impact of schema changes in UltraLite” on page 15](#).

Side effects

- ◆ Automatic commit.

Example

The following example creates a two-column index on the employee table.

```
CREATE INDEX employee_name_index
ON employee
( emp_lname, emp_fname )
```

The following example creates an index on the sales_order_items table for the prod_id column.

```
CREATE INDEX item_prod
ON sales_order_items
( prod_id )
```

See also

- ◆ [“DROP INDEX statement” on page 337](#)
- ◆ [“Working with UltraLite indexes” on page 65](#)

CREATE PUBLICATION statement

Description

Use this statement to create a publication.

Syntax

```
CREATE PUBLICATION publication-name  
(TABLE table-name [ WHERE search-condition ], ... )
```

Parameters

WHERE clause If a WHERE clause is specified, only rows satisfying *search-condition* are considered for upload from the associated table during synchronization. For information about search conditions, see [“Search conditions in UltraLite” on page 221](#).

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 establishes tables that are synchronized during a single synchronization operation. They determine 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. 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.

Example

The following statement publishes all the columns and rows of two tables.

```
CREATE PUBLICATION pub_contact (  
    TABLE contact,  
    TABLE company  
)
```

The following statement publishes only the active customer rows by including a WHERE clause that tests the status column of the customer table.

```
CREATE PUBLICATION pub_customer (  
    TABLE customer  
    WHERE status = 'active'  
)
```

See also

- ◆ [“UltraLite Clients” \[MobiLink - Client Administration\]](#)
- ◆ [“DROP PUBLICATION statement” on page 338](#)
- ◆ [“ALTER PUBLICATION statement” on page 326](#)
- ◆ [“Search conditions in UltraLite” on page 221](#)

CREATE TABLE statement

Description

Use this statement to create a new table in the database.

Syntax

```

CREATE TABLE table-name
( { column-definition | table-constraint }, ... )

column-definition :
column-name data-type [ [ NOT ]
NULL ] [ DEFAULT column-default ] [ column-constraint ... ]

column-constraint :
[ UNIQUE | PRIMARY KEY ]

table-constraint :
[ CONSTRAINT constraint-name ] spec

column-default :
{ GLOBAL AUTOINCREMENT [ ( number ) ] |
AUTOINCREMENT | CURRENT DATE |
CURRENT TIME | CURRENT TIMESTAMP |
NULL |
NEWID( ) |
constant-value }

spec :
{ PRIMARY KEY ( ordered-column-list |
[ NOT NULL ] FOREIGN KEY [ role-name ]( ordered-column-list )
REFERENCES table-name ( column-name, . . . )
[ CHECK ON COMMIT ] |
UNIQUE ( ordered-column-list ) } [ WITH MAX HASH SIZE number ]

ordered-column-list :
[ ( column-name [ ASC | DESC ] ..., ) ]

```

Parameters

column-definition Define a column in the 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. For more information on identifiers, see [“Identifiers in UltraLite” on page 203](#).
- ◆ **data-type** For information on data types, see [“Data types in UltraLite” on page 212](#).
- ◆ **[NOT] NULL** If NOT NULL is specified, or if the column is in a UNIQUE or PRIMARY KEY constraint, the column cannot contain NULL in any row. Otherwise, NULL is allowed.
- ◆ **column-default and column-constraint** Clauses that contain several parameters that are used to create an expression. These parameters are explained in [“Expressions in UltraLite” on page 215](#).

column-constraint A short form for the equivalent table constraint, with only the current row. A column constraint can be one of:

- ◆ **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.
- ◆ **PRIMARY KEY** This is the same as a unique constraint, except that a column can have only one primary key constraint. The primary key usually identifies the best identifier for a row. For example, the customer number might be the primary key for the customer table.

Columns included in primary keys are automatically made NOT NULL.

table-constraint A table constraint restricts the values that one or more columns in the table can hold in order to help ensure the integrity of data in the database. If a statement would cause a violation of a constraint, execution of the statement does not complete, any changes made by the statement before error detection are rolled back, and an error is reported.

column-default 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:

- ◆ **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. You should 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. For details on how partitioning works, see [“Using GLOBAL AUTOINCREMENT in UltraLite” \[MobiLink - Client Administration\]](#).

Each partition contains the same number of values. You assign each copy of the database a unique global database identification number. See [“global_id option” on page 104](#).

UltraLite supplies default values in a database only from the partition uniquely identified by that database's number.

- ◆ **NULL** The column can contain NULL.
- ◆ **NEWID()** A function that generates a unique identifier value. For details, see [“NEWID function \[Miscellaneous\]” on page 287](#).
- ◆ **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 209](#).

- ◆ **CURRENT DATE** CURRENT DATE returns the current year, month, and day. See [“CURRENT DATE special value” on page 208](#).
- ◆ **CURRENT TIME** The current hour, minute, second and fraction of a second. See [“CURRENT TIME special value” on page 208](#).
- ◆ **constant-value** A constant for the data type of the column. Typically the constant is a number or a string.

spec Describes additional specifications. *spec* can be one of:

- ◆ **Primary key** The primary key usually identifies the collection of columns to immediately identify the rows in a table. Columns included in primary keys cannot allow NULL.
- ◆ **Foreign key** Restricts the values for a set of columns to match the values in a primary key or, less commonly, a unique constraint of another table (the primary table).
- ◆ **role-name** 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.
- ◆ **NOT NULL** Disallow NULL 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.

- ◆ **CHECK ON COMMIT** Causes the database server to wait for a COMMIT before checking that foreign keys are enforced.

This means that database changes can be applied in any order. Otherwise, the primary key (or values for UNIQUE constraint) must be in the database before a row with those foreign key values can be added.

- ◆ **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.
- ◆ **ordered-column-list** An ordered list of columns. The ordered list can be sorted in ascending or descending order.
- ◆ **WITH MAX HASH SIZE** Sets the default index hash size in bytes. If you do not set this value, a default size of 8 bytes is used for index hashing. See [“max_hash_size property” on page 106](#).

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 would cause a violation of a constraint, execution of the statement does not complete, any changes made by the statement before error detection are undone, and an error is reported.

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 [“Impact of schema changes in UltraLite” on page 15](#).

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 [“Dispose method” \[UltraLite - .NET Programming\]](#).

Side effects

- ◆ Automatic commit.

Example

The following example creates a table for a library database to hold book information.

```
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 example 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.

```
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 example creates tables for a sales database to hold order and order item information.

```
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)  
)
```

See also

- ◆ [“DROP TABLE statement” on page 339](#)
- ◆ [“CREATE TABLE statement” \[*SQL Anywhere Server - SQL Reference*\]](#)
- ◆ [“Data types in UltraLite” on page 212](#)
- ◆ [“Overriding partition sizes for autoincremented columns” \[*MobiLink - Client Administration*\]](#)

DELETE statement

Description

Use this statement to delete rows from a table in the database.

Syntax

```
DELETE
[ FROM ] table-name
[ WHERE search-condition ]
```

Parameters

WHERE clause If a WHERE clause is specified, only rows satisfying *search-condition* are deleted. For information about search conditions, see [“Search conditions in UltraLite” on page 221](#).

Additionally, 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 DELETE statement deletes all the rows that satisfy the search condition from the named table.

The way in which UltraLite traces row states is unique. Be sure you understand the implication of deletes and row states. See [“How UltraLite tracks row states” on page 18](#).

Example

The following statement removes employee 105 from the database.

```
DELETE
FROM employee
WHERE emp_id = 105
```

The following statement removes all data prior to the year 2000 from the fin_data table.

```
DELETE
FROM fin_data
WHERE year < 2000
```

See also

- ◆ [“START SYNCHRONIZATION DELETE statement” on page 347](#)
- ◆ [“STOP SYNCHRONIZATION DELETE statement” on page 348](#)

DROP INDEX statement

Description

Use this statement to permanently remove an index definition from the database.

Syntax

```
DROP INDEX [ table-name. ]index-name
```

Remarks

You cannot drop the primary index.

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.

For UltraLite.NET users: You cannot execute this statement, unless you also call the Dispose method for all data objects (for example, ULDataReader). See [“Dispose method” \[UltraLite - .NET Programming\]](#).

Statements are not released if schema changes are initiated at the same time. See [“Impact of schema changes in UltraLite” on page 15](#).

See also

- ◆ [“CREATE INDEX statement” on page 328](#)
- ◆ [“Working with UltraLite indexes” on page 65](#)

DROP PUBLICATION statement

Description

Use this statement to drop a publication.

Syntax

```
DROP PUBLICATION publication-name1, ..., publication-nameX
```

Remarks

Use this when a publication that describes which data is uploaded to the MobiLink server has become outdated.

See also

- ◆ [“Designing synchronization in UltraLite” \[MobiLink - Client Administration\]](#)
- ◆ [“ALTER PUBLICATION statement” on page 326](#)
- ◆ [“CREATE PUBLICATION statement” on page 330](#)

Example

The following statement drops the `pub_contact` publication.

```
DROP PUBLICATION pub_contact
```

DROP TABLE statement

Description

Use this statement to permanently remove a table definition and all data in the table from a database.

Syntax

DROP TABLE *table-name*

Remarks

The DROP TABLE statement removes the definition of the indicated table. All data in the table is automatically deleted as part of the dropping process. Also, all indexes and keys for the table are dropped by the DROP TABLE statement.

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 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 [“Dispose method” \[UltraLite - .NET Programming\]](#).

Statements are not released if schema changes are initiated at the same time. See [“Impact of schema changes in UltraLite” on page 15](#).

See also

- ◆ [“ALTER TABLE statement” on page 322](#)
- ◆ [“CREATE TABLE statement” on page 331](#)

INSERT statement

Description

Use this statement to insert a single row into a table or to insert rows from a query result set.

Syntax

```
INSERT [ INTO ]
  table-name [ ( column-name, ... ) ]
  { VALUES ( expression, ... ) | SELECT statement }
```

Remarks

If the optional list of column names is given, the values are inserted one for one into the specified columns. If the list of column names is not specified, the values are inserted into the table columns in the order they were created (the same order as retrieved with SELECT *). The row is inserted into the table at an arbitrary position.

You must use either the VALUES expression or SELECT statement. You cannot use both.

Character strings inserted into tables are always stored in the same case as they are entered, regardless of whether the database is case sensitive.

Examples

The following statement adds an Eastern Sales department to the database.

```
INSERT
  INTO department ( dept_id, dept_name )
  VALUES ( 230, 'Eastern Sales' )
```

See also

- ◆ [“SELECT statement” on page 342](#)

ROLLBACK statement

Description

Use this statement to end a transaction and undo any changes made since the last COMMIT or ROLLBACK.

Syntax

```
ROLLBACK [ WORK ]
```

Parameters

WORK is an optional keyword.

Remarks

A transaction is the inserts, updates, and deletes performed on one database connection to a database between COMMIT or ROLLBACK statements. The ROLLBACK statement ends the current transaction and undoes all changes made to the database since the previous COMMIT or ROLLBACK.

See also

- ◆ [“COMMIT statement” on page 327](#)

SELECT statement

Description

Use this statement to retrieve information from the database.

Syntax

```
SELECT [ DISTINCT ] [ FIRST | TOP n [ START AT m ] ]  
  select-list  
 [ FROM table-expression, ... table-expression ]  
 [ WHERE search-condition ]  
 [ GROUP BY group-by-expression,...group-by-expression ]  
 [ HAVING search-condition ]  
 [ ORDER BY order-by-expression,...order-by-expression ]  
 [ FOR { UPDATE | READ ONLY } ]  
 [ OPTION ( FORCE ORDER ) ]
```

```
select-list :  
{ column-name | column-expression } [ AS ]  
alias-name
```

```
order-by-expression :  
{ integer | expression } [ ASC | DESC ]
```

Parameters

select-list A list of expressions, separated by commas, specifying what will be retrieved from the database. Use an asterisk (*) to select all columns of all tables in the FROM clause.

You can define an alias name following the expression in the *select-list* to represent that expression. The alias name can then be used elsewhere in the query, such as in the WHERE clause or ORDER BY clause.

DISTINCT clause If you do not specify DISTINCT, all rows that satisfy the clauses of the SELECT statement are returned. If DISTINCT is specified, duplicate output rows are eliminated. Many statements take significantly longer to execute when DISTINCT is specified, so you should reserve DISTINCT for cases where it is necessary.

FIRST or TOP or START AT clause You can explicitly retrieve only the first row of a result set or the first *n* rows of a result set. The TOP and START AT clauses provide additional flexibility in queries by explicitly limiting the result set by parameterizing it with specific instructions: the TOP clause specifies the number of rows to return and the START AT clause sets the starting point result set.

FROM clause Rows are retrieved from the tables and views specified in the *table-expression*. See [“FROM clause” on page 344](#).

WHERE clause If a WHERE clause is specified, only rows satisfying *search-condition* are selected. For information about search conditions, see [“Search conditions in UltraLite” on page 221](#).

GROUP BY clause The result of the query contains one row for each distinct set of values in the GROUP BY expressions. 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.

HAVING clause This clause selects rows based on the group values and not on the individual row values. The HAVING expressions can only be used if either the statement has a GROUP BY clause or the select list consists solely of aggregate functions. The search condition must be an aggregate expression. It can only involve aggregate expressions and expressions occurring in the GROUP BY clause. The HAVING condition is tested after a candidate row has been completely grouped.

ORDER BY clause This clause sorts the results of a query according to the expression specified in the clause. Each item in the ORDER BY list can be labeled as ASC for ascending order (the default) or DESC for descending order. If the expression is an integer n , then the query results are sorted by the n th item in the select list.

The only way to ensure that rows are returned in a particular order is to use ORDER BY. In the absence of an ORDER BY clause, UltraLite returns rows in whatever order is most efficient.

FOR clause This clause has two variations that control the query's behavior:

- ◆ **For READ ONLY** This clause prevents the query from being used for updates. You should include this clause whenever possible, since it optimizes performance.
- ◆ **FOR UPDATE** This clause is applied by default. It allows the query to be used for updates.

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 query access plans to optimize queries. For more information, see [“Query access plans in UltraLite” on page 232](#).

Example

The following statement selects the number of employees from the employee table.

```
SELECT COUNT(*)  
FROM employee
```

The following statement selects 10 rows from the employee table starting from the 40th row and ending at the 49th row.

```
SELECT TOP 10 START AT 40 * FROM employee
```

See also

- ◆ “SELECT statement” [*SQL Anywhere Server - SQL Reference*]
- ◆ “Queries: Selecting Data from a Table” [*SQL Anywhere Server - SQL Usage*]

FROM clause

Description

Use this clause to specify the database tables or views involved in a SELECT statement. When there is no FROM clause, the expressions in the SELECT statement must be constant expressions.

Syntax

FROM *table-expression*, ...

table-expression :

[*table-name*

[[**AS**] *correlation-name*]

| (*SELECT-expression*)

[**AS**] *derived-table-name* (*column-name*, ...)

| (*table-expression*)

| *table-expression* *join-operator* *table-expression* [**ON** *search-condition*]

join-operator :

, | **CROSS JOIN** | **INNER 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 an object elsewhere in the statement.

If the same correlation name is used twice for the same table in a table expression, that table is treated as if it were listed only once. For example, the following two SELECT statements are equivalent:

```
SELECT *
FROM sales_order
CROSS JOIN sales_order_items,
sales_order
CROSS JOIN employee
```

```
SELECT *
FROM sales_order
CROSS JOIN sales_order_items
CROSS JOIN employee
```

Whereas the following would be treated as two instances of the Person table, with different correlation names HUSBAND and WIFE:

```
SELECT *
FROM Person HUSBAND, Person WIFE
```

derived-table-name A derived table is a table expression that you specify as a SELECT statement. Following the parenthesized SELECT statement, is the name of the derived table and the parenthesized list of derived column names. You must include a derived column name for each select expression in the derived table.

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 table names within the SELECT statement. These references are sometimes called **inner references**. See [“Subqueries in expressions” on page 219](#).

Remarks

Derived tables

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 component tables are in the result set, and the number of combinations is usually reduced by JOIN conditions and/or WHERE conditions.

The join operator connects two tables based on common column names. Supported operators in UltraLite are:

- ◆ ,
- ◆ **CROSS JOIN**
- ◆ **INNER JOIN**
- ◆ **LEFT OUTER JOIN**
- ◆ **JOIN**

These operators can take specific conditions as specified above. The ON condition is specified for a single join operation and indicates how the join is to create rows in the result set. The JOIN operator always requires an ON condition.

A WHERE clause is used to restrict the rows in the result set, after potential rows have been created by a join.

Comma joins are the same as a CROSS JOIN. You cannot use an ON phrase with either this operator.

For INNER joins restricting with an ON or WHERE is equivalent. For OUTER joins, they are not equivalent.

For more information about joins, see [“Joins: Retrieving Data from Several Tables” \[SQL Anywhere Server - SQL Usage\]](#). Note that UltraLite does not support KEY JOINS nor NATURAL joins.

Example

The following are valid FROM clauses:

```

...
FROM employee
...

...
FROM employee NATURAL JOIN department
...

...
FROM customer
CROSS JOIN sales_order
CROSS JOIN sales_order_items
CROSS JOIN product
...

```

The following query illustrates how to use derived tables in a query:

```
SELECT lname, fname, number_of_orders
FROM customer JOIN
  ( SELECT cust_id, COUNT(*)
    FROM sales_order
    GROUP BY cust_id )
  AS sales_order_counts( cust_id,
                        number_of_orders )
ON ( customer.id = sales_order_counts.cust_id )
WHERE number_of_orders > 3
```

See also

- ◆ [“DELETE statement” on page 336](#)
- ◆ [“SELECT statement” on page 342](#)
- ◆ [“UPDATE statement” on page 352](#)
- ◆ [“Joins: Retrieving Data from Several Tables” \[*SQL Anywhere Server - SQL Usage*\]](#)

START SYNCHRONIZATION DELETE statement

Description

Use this statement to restart logging of deletes for MobiLink synchronization.

Syntax

```
START SYNCHRONIZATION DELETE
```

Remarks

UltraLite databases automatically track changes made to rows that need to be synchronized. UltraLite uploads these changes to the consolidated database during the next synchronization of the table that was changed. This statement allows you to temporarily restart tracking of deleted rows.

When a STOP SYNCHRONIZATION DELETE statement is executed, none of the delete operations executed on that connection are synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed. Repeating STOP SYNCHRONIZATION DELETE has no additional effect.

A single START SYNCHRONIZATION DELETE statement restarts the logging, regardless of the number of STOP SYNCHRONIZATION DELETE statements preceding it.

Do not use START SYNCHRONIZATION DELETE if your application does not synchronize data.

The way in which UltraLite traces row states is unique. Be sure you understand the implication of deletes and row states. See [“How UltraLite tracks row states” on page 18](#).

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;
```

See also

- ◆ [“STOP SYNCHRONIZATION DELETE statement” on page 348](#)

STOP SYNCHRONIZATION DELETE statement

Description

Use this statement to temporarily stop logging of deletes for MobiLink synchronization.

Syntax

STOP SYNCHRONIZATION DELETE

Remarks

UltraLite databases automatically track 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 temporarily suspend tracking of deleted rows.

When a STOP SYNCHRONIZATION DELETE statement is executed, none of the subsequent delete operations executed on that connection are synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed.

Repeating STOP SYNCHRONIZATION DELETE has no additional effect. A single START SYNCHRONIZATION DELETE statement restarts the logging, regardless of the number of STOP SYNCHRONIZATION DELETE statements preceding it.

This command can be useful to make corrections to a remote database, but should be used with caution as it effectively disables MobiLink synchronization.

Do not use STOP SYNCHRONIZATION DELETE if your application does not synchronize data.

The way in which UltraLite traces row states is unique. Be sure you understand the implication of deletes and row states. See [“How UltraLite tracks row states” on page 18](#).

See also

- ◆ [“START SYNCHRONIZATION DELETE statement” on page 347](#)

TRUNCATE TABLE statement

Description

Use this statement to delete all rows from a table, without deleting the table definition.

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 synchronizations. 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 and all of the indexes continue to exist until you issue a DROP TABLE statement. The column definitions and constraints remain intact.

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 [“Dispose method” \[UltraLite - .NET Programming\]](#).

Schema changes Statements are not released if schema changes are initiated at the same time. See [“Impact of schema changes in UltraLite” on page 15](#).

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. However, they do remain accessible from other connections. Use ROLLBACK to make them re-accessible to the current user. Use COMMIT to physically delete the rows, and thereby make 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.

Example

Delete all rows from the Departments table.

```
TRUNCATE TABLE Departments
```

See also

- ◆ [“DELETE statement” on page 336](#)
- ◆ [“START SYNCHRONIZATION DELETE statement” on page 347](#)
- ◆ [“STOP SYNCHRONIZATION DELETE statement” on page 348](#)

UNION operation

Description

Use this statement to combine the results of two or more select statements.

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. The component SELECT statements must each have the same number of items in the select list, and cannot contain an ORDER BY clause.

The results of UNION ALL are the combined results of the component SELECT statements. The results of UNION are the same as UNION ALL, except that duplicate rows are eliminated. Eliminating duplicates requires extra processing, so UNION ALL should be used instead of UNION where possible. UNION DISTINCT is identical to UNION.

If corresponding items in two select lists have different data types, UltraLite chooses a data type for the corresponding column in the result and automatically converts the columns in each component SELECT statement appropriately.

The column names displayed are the same column names that are displayed for the first SELECT statement. An alternative way of customizing result set column names is to use the WITH clause in the SELECT statement.

The ORDER BY clause uses integers to establish the ordering, where the integer indicates the query expression(s) on which to sort the results.

Example

The following example lists all distinct surnames of employees and customers.

```
SELECT emp_lname  
FROM Employee  
UNION  
SELECT lname  
FROM Customer
```

See also

- ◆ [“SELECT statement” on page 342](#)

UPDATE statement

Description

Use this statement to modify existing rows in database tables.

Syntax

```
UPDATE table-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.

SET clause Each named column is set to the value of the expression on the right-hand 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. For information about search conditions, see [“Search conditions in UltraLite” on page 221](#).

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.

Example

The following example transfers employee Philip Chin from the sales department (department 129) to the marketing department (department 400).

```
UPDATE employee
SET dept_id = 400
WHERE emp_id = 129
```

See also

- ◆ [“INSERT statement” on page 340](#)
- ◆ [“DELETE statement” on page 336](#)
- ◆ [“Search conditions in UltraLite” on page 221](#)

Index

Symbols

- % operator
 - modulo function, UltraLite, 284
- &
 - bitwise operator for UltraLite, 229
- a option
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite initialize database [ulinit] utility, 172
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite synchronization [ulsync] utility, 178
- b option
 - UltraLite unload data to XML [ulunload] utility, 180
 - UltraLite unload old database [ulunloadold] utility, 183
- c option
 - Interactive SQL [dbisql] utility for UltraLite, 155
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite information [ulinfo] utility, 169
 - UltraLite initialize database [ulinit] utility, 172
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite synchronization [ulsync] utility, 178
 - UltraLite unload data to XML [ulunload] utility, 180
 - UltraLite unload old database [ulunloadold] utility, 183
- codepage option
 - Interactive SQL [dbisql] utility for UltraLite, 155
- d option
 - Interactive SQL [dbisql] utility for UltraLite, 155
 - UltraLite AppForge registration [ulafreg] utility, 161
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite unload data to XML [ulunload] utility, 180
- d1 option
 - Interactive SQL [dbisql] utility for UltraLite, 155
- e option
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite synchronization [ulsync] utility, 178
 - UltraLite unload data to XML [ulunload] utility, 180
- f option
 - Interactive SQL [dbisql] utility for UltraLite, 155
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite unload data to XML [ulunload] utility, 180
 - UltraLite unload old database [ulunloadold] utility, 183
- g option
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite information [ulinfo] utility, 169
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
- h option
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
- I option
 - UltraLite load XML to database [ulload] utility, 174
- k option
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite synchronization [ulsync] utility, 178
- l option
 - UltraLite database creation [ulcreate] utility, 165
- n option
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite initialize database [ulinit] utility, 172
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite synchronization [ulsync] utility, 178
 - UltraLite unload data to XML [ulunload] utility, 180
- nogui option
 - Interactive SQL [dbisql] utility for UltraLite, 155
- o extended options
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite initialize database [ulinit] utility, 172
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite overview, 186
 - UltraLite usage, 186

- o option
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
- ol option
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite load XML to database [ulload] utility, 174
- onerror option
 - Interactive SQL [dbisql] utility for UltraLite, 155
 - UltraLite load XML to database [ulload] utility, 174
- or option
 - UltraLite information [ulinfo] utility, 169
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite synchronization [ulsync] utility, 178
 - UltraLite unload data to XML [ulunload] utility, 180
- p option
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite initialize database [ulinit] utility, 172
 - UltraLite load XML to database [ulload] utility, 174
- q option
 - Interactive SQL [dbisql] utility for UltraLite, 155
 - UltraLite AppForge registration [ulafreg] utility, 161
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite information [ulinfo] utility, 169
 - UltraLite initialize database [ulinit] utility, 172
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite synchronization [ulsync] utility, 178
 - UltraLite unload data to XML [ulunload] utility, 180
 - UltraLite unload old database [ulunloadold] utility, 183
- qq option
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
- r option
 - UltraLite AppForge registration [ulafreg] utility, 161
 - UltraLite information [ulinfo] utility, 169
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite synchronization [ulsync] utility, 178
- rc option
 - UltraLite AppForge registration [ulafreg] utility, 161
 - UltraLite information [ulinfo] utility, 169
- s option
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite unload data to XML [ulunload] utility, 180
- t option
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite unload data to XML [ulunload] utility, 180
- u option
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
- v option
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite information [ulinfo] utility, 169
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite synchronization [ulsync] utility, 178
 - UltraLite unload data to XML [ulunload] utility, 180
 - UltraLite unload old database [ulunloadold] utility, 183
- w option
 - UltraLite initialize database [ulinit] utility, 172
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
- x option
 - Interactive SQL [dbisql] utility for UltraLite, 155
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite synchronization [ulsync] utility, 178
 - UltraLite unload data to XML [ulunload] utility, 180
- y option
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite load XML to database [ulload] utility, 174

- UltraLite unload data to XML [ulunload] utility, 180
- UltraLite unload old database [ulunloadold] utility, 183
- z option
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
- .NET
 - UltraLite engine support, 11
- .NET compatibility
 - UltraLite driver for ADO.NET, 13
- /* comment indicator
 - UltraLite about, 205
- // comment indicator
 - UltraLite about, 205
- ?
 - UltraLite input parameter, 220
- @ option
 - Interactive SQL [dbisql] utility for UltraLite, 155
- @@identity global variable
 - UltraLite usage, 231
- ^
 - bitwise operator for UltraLite, 229
- |
 - bitwise operator for UltraLite, 229
- ~
 - bitwise operator for UltraLite, 229
- comment indicator
 - UltraLite about, 205

A

- ABS function
 - UltraLite SQL syntax, 241
- ACOS function
 - UltraLite SQL syntax, 241
- ActiveSync
 - supported versions, 12
- adding
 - UltraLite column methods, 58
 - UltraLite columns, 322
 - UltraLite indexes, 67
 - UltraLite users, 76
- adding a column to an UltraLite table
 - about, 58
- adding a new UltraLite user
 - about, 76
- adding UltraLite indexes
 - about, 67
- ADO.NET
 - UltraLite drivers for, 13
- AES encryption algorithm
 - UltraLite fips usage, 38
 - UltraLite reference, 102
 - UltraLite usage, 38
- aggregate expressions
 - UltraLite SQL syntax, 218
- aggregate functions
 - UltraLite alphabetical list, 236
- algorithms
 - UltraLite cryptographic module for AES FIPS encryption, 102
- aliases
 - UltraLite columns, 342
 - UltraLite DELETE statement, 336
 - UltraLite equivalents of, 214
- ALL conditions
 - UltraLite SQL, 224
- allsync tables
 - UltraLite overview, 57
- ALTER PUBLICATION statement
 - UltraLite SQL syntax, 326
- ALTER TABLE statement
 - UltraLite SQL syntax, 322
 - UltraLite usage, 59
- altering
 - UltraLite column methods, 59
 - UltraLite columns methods, 59
 - UltraLite publications, 326
 - UltraLite table methods, 59
 - UltraLite tables , 322
- altering UltraLite column definitions
 - about, 59
- ambiguous string to date conversions
 - UltraLite reference, 109
 - UltraLite usage, 35
- AND
 - bitwise operators for UltraLite, 229
 - logical operators for UltraLite, 223
- ANY conditions
 - UltraLite SQL, 225
- APIs
 - UltraLite choices, 13
 - UltraLite engine support, 11
- AppForge
 - UltraLite engine support, 11

- UltraLite support, 13
- AppForge registry update utility
 - syntax, 161
- applications, vii
 - (see also UltraLite applications)
- arc-cosine function
 - UltraLite ACOS function, 241
- arc-sine function
 - UltraLite ASIN function, 243
- arc-tangent function
 - UltraLite ATAN function, 244
- ARGN function
 - UltraLite SQL syntax, 242
- arithmetic
 - operators and UltraLite SQL syntax, 228
- arithmetic operators
 - UltraLite SQL syntax, 228
- articles
 - UltraLite copying method, 63
- ASCII
 - UltraLite sorting, 31
 - UltraLite syntax, 242
- ASIN function
 - UltraLite SQL syntax, 243
- assembling parameters into connection strings
 - UltraLite about, 49
- ATAN function
 - UltraLite SQL syntax, 244
- ATAN2 function
 - UltraLite SQL syntax, 244
- authentication
 - UltraLite bypassing, 45
 - UltraLite setup, 45
- autocommit
 - UltraLite, 19
- AUTOINCREMENT
 - @@identity (UltraLite), 231
 - UltraLite SQL syntax, 331
- average function
 - UltraLite AVG function, 245
- AVG function
 - UltraLite SQL syntax, 245
- B**
- backups
 - UltraLite databases, 19
 - UltraLite databases on Palm, 185
 - UltraLite databases on Windows CE, 51
- base 10 logarithm
 - UltraLite LOG10 function, 279
- BETWEEN conditions
 - UltraLite SQL, 226
- BIGINT data type
 - UltraLite, 212
- binary
 - UltraLite sorting, 31
- BINARY data types
 - UltraLite, 212
- bitwise operators
 - UltraLite SQL syntax, 229
- browsing
 - UltraLite table methods, 61
- browsing the information in UltraLite tables
 - about, 61
- BYTE_LENGTH function
 - UltraLite SQL syntax, 246
- BYTE_SUBSTR function
 - UltraLite SQL syntax, 246
- C**
- C++ applications
 - UltraLite engine support, 11
- C-language programming
 - UltraLite support, 13
- cache size
 - UltraLite usage, 36
- CACHE_SIZE connection parameter
 - UltraLite syntax, 128
- cascading deletes
 - not supported in UltraLite, 5
- cascading updates
 - not supported in UltraLite, 5
- case database property
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite reference, 94
 - UltraLite usage, 33
- CASE expression
 - UltraLite NULLIF function, 288
 - UltraLite SQL syntax, 217
- case sensitivity
 - comparison operators for UltraLite, 222
 - UltraLite case reference, 94
 - UltraLite strings, 204
- case sensitivity considerations

-
- UltraLite about, 33
 - CAST function
 - UltraLite SQL syntax, 247
 - catalog
 - UltraLite system tables, 192
 - CE_FILE connection parameter
 - UltraLite syntax, 133
 - CEILING function
 - UltraLite SQL syntax, 248
 - Certicom
 - UltraLite cryptographic module, 38
 - CHAR data type
 - UltraLite, 212
 - CHAR function
 - UltraLite SQL syntax, 249
 - CHAR_LENGTH function
 - UltraLite SQL syntax, 250
 - character functions
 - UltraLite alphabetical list, 239
 - character set considerations
 - UltraLite, 31
 - character sets
 - synchronization for UltraLite, 31
 - UltraLite databases, 32
 - UltraLite on Palm OS, 32
 - UltraLite on Symbian, 32
 - UltraLite on Windows, 32
 - UltraLite on Windows CE, 32
 - UltraLite strings, 204
 - character strings
 - UltraLite embedded SQL, 158
 - CHARINDEX function
 - UltraLite SQL syntax, 249
 - check constraints
 - UltraLite limitations, 5
 - checksum_level
 - UltraLite reference, 95
 - checksums
 - UltraLite checksum_level reference, 95
 - UltraLite usage, 37
 - choosing a data management component
 - UltraLite about, 11
 - choosing an index type
 - UltraLite about, 66
 - choosing an UltraLite deployment environment
 - about, 12
 - choosing creation-time database properties
 - about, 30
 - choosing your programming interface
 - UltraLite about, 12
 - client
 - embedded engine for UltraLite, 11
 - COALESCE function
 - UltraLite SQL syntax, 251
 - code points
 - UltraLite, 31
 - collation sequences
 - UltraLite about, 31
 - UltraLite changing, 31
 - column compression
 - UltraLite SQL ALTER TABLE statement, 322
 - column names
 - UltraLite SQL syntax, 216
 - column names in expressions
 - UltraLite about, 216
 - columns
 - UltraLite adding methods, 58
 - UltraLite aliases, 342
 - UltraLite altering, 322
 - UltraLite altering methods, 59
 - UltraLite altering usage, 59
 - UltraLite copying method, 63
 - UltraLite renaming, 322
 - combining
 - UltraLite result of multiple select statements, 351
 - comma-separated lists
 - UltraLite LIST function syntax, 277
 - command line utilities
 - Interactive SQL [dbisql] syntax for UltraLite, 155
 - UltraLite AppForge registration [ulafreg] utility, 161
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite engine start [uleng10] utility, 168
 - UltraLite engine stop [ulstop] utility, 177
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite information [ulinfo] utility, 169
 - UltraLite initialize database [ulinit] utility, 172
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite Palm [ULDBUtil] utility, 185
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite synchronization [ulsync] , 178
 - UltraLite unload data to XML [ulunload] utility, 180

- UltraLite unload old database [ulunloadold] utility, 183
- comments
 - UltraLite syntax, 205
- comments in UltraLite
 - about, 205
- COMMIT statement
 - UltraLite SQL syntax, 327
- committing
 - UltraLite COMMIT syntax, 327
 - UltraLite database rows, 18
 - UltraLite transaction overview , 19
- comparing
 - UltraLite and SQL Anywhere databases, 5
 - UltraLite statistics, 8
- comparing UltraLite and SQL Anywhere
 - about, 5
- comparison operators
 - dynamic SQL syntax for UltraLite, 222
 - UltraLite SQL, 222
- compatibility
 - UltraLite SQL, 222
- compressed columns
 - UltraLite SQL ALTER TABLE statement, 322
- computed columns
 - UltraLite limitations, 5
- CON connection parameter
 - UltraLite syntax, 130
- concatenating strings
 - string operators for UltraLite, 229
- concurrency
 - synchronizing UltraLite applications, 18
 - UltraLite databases, 17
- concurrent access
 - UltraLite engine, 11
- conditions
 - ALL conditions for UltraLite SQL, 224
 - ANY for UltraLite SQL, 225
 - BETWEEN for UltraLite SQL, 226
 - EXISTS for UltraLite SQL, 226
 - IN for UltraLite SQL, 227
 - searching in UltraLite SQL, 221
- conduit
 - installing, 163
 - installing for CustDB, 163
- configuring post-creation database options
 - about, 40
- connecting
 - UltraLite database troubleshooting, 49
 - UltraLite databases, 45
 - connecting to an UltraLite database
 - about, 43
 - connection failures
 - UltraLite troubleshooting, 47
 - connection methods
 - UltraLite about, 44
 - connection parameters
 - DBN for UltraLite, 142
 - UltraLite, 43
 - UltraLite CACHE_SIZE , 128
 - UltraLite CE_FILE, 133
 - UltraLite choosing between, 50
 - UltraLite CON, 130
 - UltraLite connection summary, 47
 - UltraLite DBF, 131
 - UltraLite DBKEY, 143
 - UltraLite file_name, 131
 - UltraLite key, 143
 - UltraLite list of, 44
 - UltraLite NT_FILE, 135
 - UltraLite overview, 47
 - UltraLite PALM_ALLOW_BACKUP, 145
 - UltraLite PALM_DB , 139
 - UltraLite PALM_FILE, 137
 - UltraLite password, 146
 - UltraLite precedence of, 49
 - UltraLite PWD, 146
 - UltraLite RESERVE_SIZE , 148
 - UltraLite START , 149
 - UltraLite supplying, 47
 - UltraLite SYMBIAN_FILE , 140
 - UltraLite troubleshooting, 47
 - UltraLite UID , 150
 - UltraLite userid , 150
 - connection strings
 - UltraLite parameters overview, 47
 - UltraLite setting , 49
 - connections
 - concurrency in UltraLite, 17
 - UltraLite limitations, 8
 - UltraLite overview, 43
 - consolidated databases
 - UltraLite sample, 88
 - constants
 - UltraLite SQL syntax, 216
 - constants in expressions

- UltraLite about, 216
- constraints
 - UltraLite ALTER TABLE statement, 322
 - UltraLite renaming, 322
- conventions
 - documentation, x
 - file names in documentation, xii
- conversion functions
 - UltraLite alphabetical list, 236
- CONVERT function
 - UltraLite SQL syntax, 252
- converting
 - UltraLite ambiguous dates, 35
 - UltraLite ambiguous dates reference, 109
- converting strings
 - UltraLite about, 239
- copying
 - UltraLite table method, 62
- copying and pasting data to/from UltraLite databases
 - about, 62
- COS function
 - UltraLite SQL syntax, 254
- cosine function
 - UltraLite COS function, 254
- COT function
 - UltraLite SQL syntax, 255
- cotangent function
 - UltraLite COT function, 255
- COUNT function
 - UltraLite SQL syntax, 255
- count operation
 - UltraLite query access plans, 233
- create database wizard
 - UltraLite, 25
 - UltraLite usage, 25
- CREATE INDEX statement
 - UltraLite SQL syntax, 328
 - UltraLite usage, 67
- CREATE PUBLICATION statement
 - UltraLite SQL syntax, 330
 - UltraLite usage, 73
- create publication wizard
 - UltraLite rows publishing, 74
 - UltraLite tables publishing, 72
- CREATE TABLE statement
 - UltraLite SQL syntax, 331
 - UltraLite usage, 57
- creating
 - reference databases for UltraLite, 26
 - UltraLite CREATE PUBLICATION statement, 330
 - UltraLite databases, 24, 165
 - UltraLite databases from a MobiLink sync model, 26
 - UltraLite databases from the command prompt, 25
 - UltraLite databases from XML, 28
 - UltraLite indexes, 67
 - UltraLite publications tables, 72
 - UltraLite table methods, 56
 - UltraLite tables, 331
 - UltraLite users, 76
- creating and configuring UltraLite databases
 - about, 23
- creating databases
 - Sybase Central UltraLite plug-in, 25
- creating UltraLite databases
 - about, 24
- creating UltraLite tables
 - about, 56
- creator IDs
 - UltraLite PALM_DB parameter, 139
- CROSS JOIN clause
 - UltraLite SQL syntax, 344
- Crossfire
 - UltraLite support, 13
- cryptology
 - UltraLite Certicom module, 38
- CURRENT DATE
 - UltraLite special value, 208
- current date function
 - UltraLite TODAY function, 310
- current row
 - concurrency in UltraLite, 17
- CURRENT TIME
 - UltraLite special value, 208
- CURRENT TIMESTAMP
 - SQL special value for UltraLite, 5
 - UltraLite special value, 209
- cursors
 - concurrency in UltraLite, 17, 18
- CustDB application
 - starting in UltraLite, 83
- CustDB UltraLite sample
 - UltraLite about, 79
 - UltraLite database synchronization, 86
 - UltraLite file locations, 81

- UltraLite source code, 81
- UltraLite tutorial, 80
- custdb.db
 - location of, 81
- custdb.sql
 - calling synchronization scripts, 88
- custdb.udb
 - UltraLite location of, 81
- D**
- data
 - UltraLite selecting rows, 342
 - UltraLite viewing methods, 61
- data management
 - UltraLite, 11
- Data Manager
 - UltraLite database storage, 50
- data type conversion functions
 - UltraLite about, 236
- data types
 - BIGINT in UltraLite, 212
 - BINARY in UltraLite, 212
 - CHAR in UltraLite, 212
 - DATE in UltraLite, 212
 - DECIMAL in UltraLite, 212
 - DOUBLE in UltraLite, 212
 - FLOAT in UltraLite, 212
 - INT in UltraLite, 212
 - INTEGER in UltraLite, 212
 - LONG BINARY in UltraLite, 212
 - LONG VARCHAR in UltraLite, 212
 - NUMERIC in UltraLite, 212
 - REAL in UltraLite, 212
 - SMALLINT in UltraLite, 212
 - TIME in UltraLite, 212
 - TIMESTAMP in UltraLite, 212
 - TINYINT in UltraLite, 212
 - UltraLite, 212
 - UltraLite alias equivalents, 214
 - UltraLite SQL, 212
 - UltraLite SQL conversion functions, 236
 - VARBINARY in UltraLite, 212
 - VARCHAR in UltraLite, 212
- data types in UltraLite
 - about, 212
- database
 - UltraLite extended options, 187
 - UltraLite synchronization utility [ulsync] syntax, 178
- database engine
 - UltraLite , 11
- database files, vii
 - (see also UltraLite databases)
 - UltraLite connection parameters, 50
 - UltraLite encrypting, 143
- database objects
 - UltraLite copying method, 63
- database options, vii, 40
 - (see also database options (UltraLite))
- database options (UltraLite)
 - browsing, 78
 - global_id reference, 104
 - global_id usage, 40
 - ml_remote_id reference, 108
 - ml_remote_id usage, 40
- database page size considerations
 - UltraLite about, 36
- database properties, vii
 - (see also database properties (UltraLite))
- database properties (UltraLite)
 - browsing, 78
 - case reference, 94
 - case usage, 33
 - checksum_level reference, 95
 - choosing creation-time, 30
 - date usage, 34
 - date_format reference, 97
 - date_order reference, 100
 - fips property reference, 102
 - fips usage, 38
 - max_hash_size reference, 106
 - max_hash_size usage, 33
 - nearest_century reference, 109
 - nearest_century usage, 35
 - obfuscate reference, 110
 - obfuscate usage, 38
 - page_size reference, 112
 - page_size usage, 36
 - precision reference, 114
 - precision usage, 36
 - scale reference, 116
 - scale usage, 36
 - time_format reference, 118
 - time_format usage, 34
 - timestamp_format reference, 120

- timestamp_format usage, 35
- timestamp_increment reference, 123
- timestamp_increment usage, 35
- utf8_encoding reference, 125
- utf8_encoding usage, 32
- database utilities
 - UltraLite database connections, 49
- databases, vii
 - (see also UltraLite databases)
 - comparing UltraLite and SQL Anywhere, 5
 - creating with UltraLite plug-in, 25
 - UltraLite inserting rows, 340
- DATALENGTH function
 - UltraLite SQL syntax, 256
- date considerations
 - UltraLite about, 34
- DATE data type
 - UltraLite, 212
- DATE function
 - UltraLite SQL syntax, 257
- date functions
 - UltraLite alphabetical list, 236
- date parts
 - about, 34, 237
- date_format database property
 - UltraLite reference, 97
 - UltraLite usage, 34
- date_order database property
 - UltraLite reference, 100
 - UltraLite usage, 34
- DATEADD function
 - UltraLite SQL syntax, 257
- DATEDIFF function
 - UltraLite SQL syntax, 258
- DATEFORMAT function
 - UltraLite SQL syntax, 259
- DATENAME function
 - UltraLite SQL syntax, 260
- DATEPART function
 - UltraLite SQL syntax, 260
- dates
 - UltraLite ambiguous string conversions, 35, 109
 - UltraLite conversion functions, 236
 - UltraLite formatting reference, 97
 - UltraLite formatting usage, 34
 - UltraLite ordering, 100
 - UltraLite rollover point, 35, 109
- dates and times in UltraLite
 - about, 211
- datetime
 - UltraLite conversion functions, 236
- DATETIME function
 - UltraLite SQL syntax, 261
- DAY function
 - UltraLite SQL syntax, 262
- day of week
 - UltraLite DOW function, 265
- DAYNAME function
 - UltraLite SQL syntax, 262
- DAYS function
 - UltraLite SQL syntax, 263
- DBF connection parameter
 - UltraLite syntax, 131
- dbisql utility
 - UltraLite exit codes, 157
 - UltraLite syntax, 155
- DBKEY connection parameter
 - UltraLite syntax, 143
- DBN connection parameter
 - UltraLite syntax, 142
- DDL statements
 - UltraLite schema changes with, 15
- DECIMAL data type
 - UltraLite, 212
- decimal point position considerations
 - UltraLite about, 36
- decimal precision
 - UltraLite precision reference, 114
 - UltraLite precision usage, 36
- decimals
 - used in UltraLite, 206
- DEFAULT TIMESTAMP columns
 - UltraLite SQL syntax, 331
- default values
 - UltraLite CURRENT DATE, 208
 - UltraLite CURRENT TIME, 208
 - UltraLite CURRENT TIMESTAMP, 209
 - UltraLite SQLCODE, 210
- defaults
 - UltraLite autoincrement, 331
- definitions
 - UltraLite altering tables , 322
- DEGREES function
 - SQL syntax, 264
- DELETE statement
 - UltraLite dynamic SQL syntax, 336

- deleting
 - UltraLite columns, 322
 - UltraLite databases, 18
 - UltraLite deleting all rows from a table, 349
 - UltraLite indexes, 68
 - UltraLite publications, 75
 - UltraLite SQL STOP SYNCHRONIZATION DELETE statement, 348
 - UltraLite START SYNCHRONIZATION DELETE statement, 347
 - UltraLite table methods, 60
 - UltraLite users, 77
 - UltraLite utility to delete databases, 185
 - deleting an existing UltraLite user
 - about, 77
 - demos, vii
 - (see also tutorials)
 - derived tables
 - subqueries for UltraLite SQL, 219
 - UltraLite FROM clause, 344
 - UltraLite SQL, 219
 - desktop creation
 - UltraLite about, 24
 - development platforms
 - UltraLite, 12
 - devices
 - UltraLite multiple connection parameters for, 50
 - DIFFERENCE function
 - UltraLite SQL syntax, 264
 - digits
 - UltraLite maximum number, 36, 114
 - DISTINCT keyword
 - UltraLite SQL, 342
 - distinct operation
 - UltraLite query access plans, 233
 - documentation
 - conventions, x
 - SQL Anywhere, viii
 - DOUBLE data type
 - UltraLite, 212
 - double hyphen
 - UltraLite comment indicator, 205
 - double slash
 - UltraLite comment indicator, 205
 - DOW function
 - UltraLite SQL syntax, 265
 - DROP INDEX statement
 - UltraLite SQL syntax, 337
 - UltraLite usage, 68
 - DROP PUBLICATION statement
 - UltraLite SQL syntax, 338
 - UltraLite usage, 75
 - DROP TABLE statement
 - UltraLite SQL syntax, 339
 - UltraLite usage, 61
 - dropping
 - UltraLite columns, 322
 - UltraLite indexes, 68
 - UltraLite SQL CREATE INDEX statement, 328
 - UltraLite SQL DROP INDEX statement, 337
 - UltraLite SQL DROP PUBLICATION statement, 338
 - UltraLite SQL DROP TABLE statement, 339
 - UltraLite table methods, 60
 - dropping a publication for UltraLite
 - about, 75
 - dropping an index
 - UltraLite about, 68
 - dropping publications
 - UltraLite clients, 75
 - dropping UltraLite tables
 - about, 60
 - dummy operation
 - UltraLite query access plans, 233
 - dynamic SQL
 - arithmetic operators for UltraLite, 228
 - bitwise operators for UltraLite, 229
 - logical operators for UltraLite, 223
 - operator precedence for UltraLite, 229
 - string operators for UltraLite, 229
- ## E
- editing
 - UltraLite table methods, 61
 - ELSE
 - CASE expression for UltraLite, 217
 - IF expressions for UltraLite, 217
 - embedded SQL
 - line numbers for UltraLite, 158
 - NULL values in UltraLite, 207
 - UltraLite character strings, 158
 - UltraLite preprocessor for, 158
 - UltraLite support, 13
 - encoding
 - UltraLite utf8, 32, 125

encryption

- UltraLite database property reference, 102
- UltraLite encryption keys, 143
- UltraLite fips reference, 38
- UltraLite obfuscate property reference, 110
- UltraLite obfuscate property usage, 38
- UltraLite reference, 102
- UltraLite usage, 38

END

- CASE expression for UltraLite, 217

ENDIF

- IF expressions for UltraLite, 217

engine

- embedded client for UltraLite, 11

entity-relationship diagrams

- UltraLite about, 63

entity-relationship tab

- UltraLite using, 63

environment variables

- ERRORLEVEL for UltraLite, 157
- UltraLite ULSQLCONNECT, 53
- UltraLite usage, 49

ER Diagram tab

- UltraLite about, 63

erasing

- UltraLite table methods, 60

error codes

- UltraLite database creation [ulcreate] utility, 189
- UltraLite database synchronization [ulsync] utility, 189
- UltraLite load XML to database [ulload] utility, 189
- UltraLite unload data to XML [ulunload] utility, 189
- UltraLite utilities, 189

ERRORLEVEL environment variable

- Interactive SQL return code for UltraLite, 157

examples, vii

- (see also tutorials)

exclusive OR

- bitwise operator for UltraLite, 229

EXISTS conditions

- UltraLite SQL, 226

exit codes

- Interactive SQL [dbisql] utility for UltraLite, 157
- UltraLite database creation [ulcreate] utility, 189
- UltraLite database synchronization [ulsync] utility, 189
- UltraLite load XML to database [ulload] utility, 189
- UltraLite unload data to XML [ulunload] utility, 189

EXP function

- UltraLite SQL syntax, 266

expansion cards

- UltraLite writing to, 51

EXPLANATION function

- UltraLite SQL syntax, 266

exploring the CustDB Samples for UltraLite

- about, 79

exponential function

- UltraLite EXP function, 266

exponents

- in UltraLite, 206

expressions

- aggregate for UltraLite, 218
- CASE expressions for UltraLite, 217
- IF expressions for UltraLite, 217
- SQL operator precedence for UltraLite, 229
- subqueries for UltraLite SQL, 219
- UltraLite column names, 216
- UltraLite constants, 216
- UltraLite input parameters, 220
- UltraLite SQL, 215

expressions in UltraLite

- about, 215

extended creation-time options

- about, 186

extended options

- about, 187
- UltraLite creation-time properties, 186
- UltraLite overview, 186
- UltraLite synchronization, 187

extended synchronization parameters

- about, 187

F

failures

- UltraLite preventing out-of-memory errors, 148

features

- UltraLite, 4

Federal Information Processing Standards (see FIPS)

feedback

- documentation, xv
- providing, xv

- fetching rows
 - concurrency in UltraLite, 18
- file names
 - UltraLite connection parameters, 50
- file objects
 - UltraLite types, 15
- file systems, vii
 - (see also VFS)
- files
 - UltraLite CustDB sample location, 81
- filter operation
 - UltraLite query access plans, 233
- filtering
 - UltraLite table methods, 62
- FIPS
 - UltraLite fips property reference, 102
 - UltraLite fips property usage, 38
 - UltraLite setup and deployment for, 102
- fips database property
 - UltraLite usage, 38
- FIRST clause
 - UltraLite SQL SELECT statement, 342
- FLOAT data type
 - UltraLite, 212
- FLOOR function
 - UltraLite SQL syntax, 267
- FOR clause
 - UltraLite SELECT statement, 343
- FORCE ORDER clause
 - UltraLite SELECT statement, 343
- foreign keys
 - UltraLite copying method, 63
 - UltraLite foreign keys, 331
 - UltraLite of unnamed, 331
- format options (UltraLite)
 - case-sensitivity reference, 94
 - case-sensitivity usage, 33
 - date_format reference, 97
 - date_format usage, 34
 - date_order reference, 100
 - date_order usage, 34
 - max_hash_size reference, 106
 - max_hash_size usage, 33
 - nearest_century reference, 109
 - nearest_century usage, 35
 - precision reference, 114
 - precision usage, 36
 - scale reference, 116
 - scale usage, 36
 - time_format reference, 118
 - time_format usage, 34
 - timestamp_format reference, 120
 - timestamp_format usage, 35
 - timestamp_increment reference, 123
 - timestamp_increment usage, 35
 - UltraLite checksum_level reference, 95
 - utf8_encoding reference, 125
 - utf8_encoding usage, 32
- FROM clause
 - UltraLite SELECT statement, 342
 - UltraLite SQL syntax, 344
- functions
 - alphabetical list of all functions, 241
 - types of function for UltraLite, 236
 - UltraLite aggregate, 236
 - UltraLite data type conversion SQL, 236
 - UltraLite date and time, 236
 - UltraLite miscellaneous, 238
 - UltraLite numeric, 239
 - UltraLite string, 239
- functions, aggregate
 - about, 236
 - UltraLite AVG, 245
 - UltraLite COUNT, 255
 - UltraLite LIST, 277
 - UltraLite MAX, 281
 - UltraLite MIN, 282
 - UltraLite SUM, 308
- functions, data type conversion
 - UltraLite about, 236
 - UltraLite CAST, 247
 - UltraLite CONVERT, 252
 - UltraLite HEXTOINT, 269
 - UltraLite INTTOHEX, 273
 - UltraLite ISDATE, 273
 - UltraLite ISNULL, 273
- functions, date and time
 - UltraLite DATE, 257
 - UltraLite DATEADD, 257
 - UltraLite DATEDIFF, 258
 - UltraLite DATEFORMAT, 259
 - UltraLite DATENAME, 260
 - UltraLite DATEPART, 260
 - UltraLite DATETIME, 261
 - UltraLite DAY, 262
 - UltraLite DAYNAME, 262

- UltraLite DAYS, 263
- UltraLite DOW, 265
- UltraLite GETDATE, 267
- UltraLite HOUR, 269
- UltraLite HOURS, 270
- UltraLite MINUTE, 282
- UltraLite MINUTES, 283
- UltraLite MONTH, 285
- UltraLite MONTHNAME, 285
- UltraLite MONTHS, 286
- UltraLite NOW, 288
- UltraLite QUARTER, 291
- UltraLite SECOND, 297
- UltraLite SECONDS, 298
- UltraLite TODAY, 310
- UltraLite WEEKS, 314
- UltraLite YEAR, 315
- UltraLite YEARS, 315
- UltraLite YMD, 316
- UltraLite, about, 236
- functions, miscellaneous
 - UltraLite about, 238
 - UltraLite ARGN, 242
 - UltraLite COALESCE, 251
 - UltraLite EXPLANATION, 266
 - UltraLite GREATER, 268
 - UltraLite IFNULL, 271
 - UltraLite LESSER, 276
 - UltraLite NEWID, 287
 - UltraLite NULLIF, 288
 - UltraLiteSHORT_PLAN, 299
- functions, numeric
 - DEGREES, 264
 - UltraLite about, 239
 - UltraLite ABS, 241
 - UltraLite ACOS, 241
 - UltraLite ASIN, 243
 - UltraLite ATAN, 244
 - UltraLite ATAN2, 244
 - UltraLite CEILING, 248
 - UltraLite COS, 254
 - UltraLite COT, 255
 - UltraLite EXP, 266
 - UltraLite FLOOR, 267
 - UltraLite LOG, 278
 - UltraLite LOG10, 279
 - UltraLite MOD, 284
 - UltraLite PI, 290
 - UltraLite POWER, 291
 - UltraLite RADIANS, 292
 - UltraLite REMAINDER, 292
 - UltraLite ROUND, 296
 - UltraLite SIGN, 300
 - UltraLite SIN, 301
 - UltraLite SQRT, 303
 - UltraLite TAN, 309
 - UltraLite TRUNCNUM, 311
- functions, string
 - UltraLite about, 239
 - UltraLite ASCII, 242
 - UltraLite BYTE_LENGTH, 246
 - UltraLite BYTE_SUBSTR, 246
 - UltraLite CHAR, 249
 - UltraLite CHAR_LENGTH, 250
 - UltraLite CHARINDEX, 249
 - UltraLite DIFFERENCE, 264
 - UltraLite INSERTSTR, 272
 - UltraLite LCASE, 274
 - UltraLite LEFT, 275
 - UltraLite LENGTH, 275
 - UltraLite LOCATE, 278
 - UltraLite LOWER, 280
 - UltraLite LTRIM, 280
 - UltraLite PATINDEX, 289
 - UltraLite REPEAT, 293
 - UltraLite REPLACE, 294
 - UltraLite REPLICATE, 295
 - UltraLite RIGHT, 295
 - UltraLite SIMILAR, 300
 - UltraLite SOUNDEX, 302
 - UltraLite SPACE, 303
 - UltraLite STR, 304
 - UltraLite STRING, 305
 - UltraLite STUFF, 306
 - UltraLite SUBSTRING, 307
 - UltraLite TRIM, 310
 - UltraLite UCASE, 311
 - UltraLite UPPER, 312
 - UltraLite UUIDTOSTR, 313
 - UltraLiteRTRIM, 297
 - UltraLiteSTRTOUUID, 305
- functions, system
 - UltraLite DATALENGTH, 256

G

GETDATE function
 UltraLite SQL syntax, 267

global autoincrement
 UltraLite global_id reference, 104

global database ID considerations
 UltraLite about, 40

global database identifier
 UltraLite global_id reference, 104

global variables
 @@identity (UltraLite), 231

global_database_id option
 UltraLite CREATE TABLE statement, 331

global_id option
 UltraLite reference, 104
 UltraLite usage, 40

globally unique identifiers
 UltraLiteSQL syntax for NEWID function, 287

government
 UltraLite security for, 38, 102

GREATER function
 UltraLite SQL syntax, 268

GROUP BY clause
 UltraLite SELECT statement, 342

group-by operation
 UltraLite query access plans, 233

GUIDs
 UltraLite SQL syntax for UUIDTOSTR function, 313
 UltraLiteSQL syntax for NEWID function, 287
 UltraLiteSQL syntax for STRTOUUID function, 305

H

hash
 UltraLite configuring size for, 71
 UltraLite optimal size for, 69
 UltraLite size considerations, 33

hashing
 UltraLite indexes reference, 106

HAVING clause
 UltraLite SELECT statement, 343

hexadecimal strings
 UltraLite about, 269

HEXTOINT function
 UltraLite SQL syntax, 269

host platforms

 UltraLite Windows supported platforms, 12

HotSync conduit
 installing, 163
 installing for CustDB, 163

HOUR function
 UltraLite SQL syntax, 269

HOURS function
 UltraLite SQL syntax, 270

I

icons
 used in manuals, xii

identifiers
 UltraLite SQL, 203

identifiers in UltraLite
 about, 203

IDENTITY column
 @@identity (UltraLite), 231

IDs
 ml_remote_id reference, 108
 ml_remote_id usage, 40
 UltraLite global database reference, 104
 UltraLite global database usage, 40
 UltraLite user, 45

IF expressions
 UltraLite SQL syntax, 217

IFNULL function
 UltraLite SQL syntax, 271

IN conditions
 UltraLite SQL, 227

in-process runtime
 UltraLite, 11

index creation wizard
 UltraLite using, 67

index performance considerations
 UltraLite about, 33

index-scan operation
 UltraLite query access plans, 233

indexes
 UltraLite copying method, 63
 UltraLite creating, 67
 UltraLite databases, 20
 UltraLite deleting, 68
 UltraLite hash considerations, 33
 UltraLite hash value reference, 106
 UltraLite non-unique indexes, 66
 UltraLite page_size usage, 36

- UltraLite performance enhancements, 69
 - UltraLite primary keys, 27
 - UltraLite types, 66
 - UltraLite unique indexes, 66, 328
 - UltraLite unique keys, 66
 - UltraLite when to create, 67
 - UltraLite when to use, 65
 - UltraLite working with, 65
 - indicators
 - UltraLite comments in SQL block, 205
 - indices
 - UltraLite system table, 194, 195
 - initializing databases
 - Sybase Central UltraLite plug-in, 25
 - INNER JOIN clause
 - UltraLite SQL syntax, 344
 - inner references
 - subqueries for UltraLite SQL, 219
 - input parameters
 - UltraLite about, 220
 - INSERT statement
 - UltraLite object copying usage, 63
 - UltraLite SQL syntax, 340
 - inserting
 - UltraLite inserting rows, 340
 - INSERTSTR function
 - UltraLite SQL syntax, 272
 - install-dir
 - documentation usage, xii
 - INT data type
 - UltraLite, 212
 - INTEGER data type
 - UltraLite, 212
 - integrity
 - UltraLite CREATE TABLE statement, 331
 - integrity constraints
 - UltraLite usage, 331
 - Interactive SQL
 - UltraLite command line, 155
 - UltraLite displaying plans, 232
 - UltraLite plan interpretation, 233
 - UltraLite plan operations, 233
 - UltraLite text plans, 232
 - Interactive SQL utility [dbisql]
 - UltraLite exit codes, 157
 - UltraLite syntax, 155
 - introducing CustDB
 - about, 80
 - introducing the UltraLite database management system
 - about, 4
 - introducing UltraLite database connections
 - about, 44
 - introduction to UltraLite utilities
 - about, 154
 - INTTOHEX function
 - UltraLite SQL syntax, 273
 - IS
 - logical operators for UltraLite, 223
 - ISDATE function
 - UltraLite SQL syntax, 273
 - ISNULL function
 - UltraLite SQL syntax, 273
 - isolation levels
 - UltraLite, 18
- J**
- join operation
 - UltraLite query access plans, 233
 - joins
 - UltraLite FROM clause syntax, 344
- K**
- key connection parameter
 - UltraLite syntax, 143
 - KEY JOIN clause
 - UltraLite SQL syntax, 344
 - keys
 - registry for ulcond cache size, 163
 - UltraLite index creation from, 66
 - UltraLite index hash, 33
 - UltraLite primary, 56
 - keyset operation
 - UltraLite query access plans, 233
 - keywords
 - UltraLite SQL, 202
 - keywords in UltraLite
 - about, 202
- L**
- LCASE function
 - UltraLite SQL syntax, 274
 - LEFT function
 - UltraLite SQL syntax, 275
 - LEFT OUTER JOIN clause
 - UltraLite SQL syntax, 344

- LENGTH function
 - UltraLite SQL syntax, 275
- LESSER function
 - UltraLite SQL syntax, 276
- libraries
 - choosing in UltraLite, 13
 - UltraLite required for FIPS, 102
- like-scan operation
 - UltraLite query access plans, 233
- limitations
 - UltraLite, 8
 - UltraLite data types, 212
- line length
 - sqlpp utility output for UltraLite, 158
- LIST function
 - UltraLite SQL syntax, 277
- lists
 - UltraLite LIST function syntax, 277
- literals
 - UltraLite about strings, 216
- loading
 - UltraLite databases, 174
- LOCATE function
 - UltraLite SQL syntax, 278
- locking
 - UltraLite concurrency, 17
- LOG function
 - UltraLite SQL syntax, 278
- LOG10 function
 - UltraLite SQL syntax, 279
- logical operators
 - UltraLite SQL syntax, 223
- logs
 - UltraLite internal mechanism, 15, 19
- lojoin operation
 - UltraLite query access plans, 233
- LONG BINARY data type
 - UltraLite, 212
- LONG VARCHAR data type
 - UltraLite, 212
- LOWER function
 - UltraLite SQL syntax, 280
- lowercase strings
 - UltraLite LCASE function, 274
 - UltraLite LOWER function, 280
- LTRIM function
 - UltraLite SQL syntax, 280

M

- M-Business Anywhere
 - UltraLite engine support, 11
 - UltraLite support, 13
- mathematical expressions
 - arithmetic operators for UltraLite, 228
- MAX function
 - UltraLite SQL syntax, 281
- max_hash_size property
 - UltraLite reference, 106
 - UltraLite usage, 33
- maximum
 - columns per table for UltraLite, 8
 - connections per UltraLite database, 8
 - rows per table for UltraLite, 8
 - tables per UltraLite database, 8
 - UltraLite date ranges, 212
- media failures
 - UltraLite databases, 19
- memory failures
 - UltraLite preventing, 148
- memory usage
 - UltraLite database storage, 50
 - UltraLite indexes, 20
 - UltraLite row states, 18
- metadata
 - UltraLite considering for reserve size, 148
- MIN function
 - UltraLite SQL syntax, 282
- minimum
 - UltraLite date ranges, 212
- MINUTE function
 - UltraLite SQL syntax, 282
- MINUTES function
 - UltraLite SQL syntax, 283
- ml_add_connection_script system procedure
 - adding, 88
- ml_add_table_script system procedure
 - adding, 88
- ml_remote_id
 - UltraLite property configuration, 169
 - UltraLite setting value of, 40
- ml_remote_id option
 - UltraLite reference, 108
 - UltraLite usage, 40
- MobiLink

-
- UltraLite CREATE PUBLICATION statement, 330
 - UltraLite SQL DROP PUBLICATION statement, 338
 - UltraLite SQL STOP SYNCHRONIZATION DELETE statement, 348
 - UltraLite START SYNCHRONIZATION DELETE statement, 347
 - UltraLite user ID uniqueness, 40
 - MobiLink synchronization
 - setting timestamp_increment reference, 123
 - MOD function
 - UltraLite SQL syntax, 284
 - modeling
 - UltraLite databases from MobiLink, 26
 - modifying
 - UltraLite columns, 322
 - UltraLite tables, 322
 - MONEY
 - UltraLite equivalent of, 214
 - MONTH function
 - UltraLite SQL syntax, 285
 - MONTHNAME function
 - UltraLite SQL syntax, 285
 - MONTHS function
 - UltraLite SQL syntax, 286
 - multi-process access
 - UltraLite engine, 11
 - multi-threaded applications
 - UltraLite, 11
 - multiple databases
 - UltraLite, 17
 - multiple devices
 - UltraLite connection parameters for, 50
- N**
- names
 - UltraLite column names, 216
 - NATURAL JOIN clause
 - UltraLite SQL syntax, 344
 - nearest century conversion considerations
 - UltraLite about, 35
 - nearest_century database property
 - UltraLite reference, 109
 - UltraLite usage, 35
 - NEWID function
 - UltraLiteSQL syntax, 287
 - newsgroups
 - technical support, xv
 - non-unique indexes
 - UltraLite index creation from, 66
 - nosync tables
 - UltraLite overview, 57
 - NOT
 - bitwise operator for UltraLite, 229
 - logical operators for UltraLite, 223
 - NOW function
 - UltraLite SQL syntax, 288
 - NT_FILE connection parameter
 - UltraLite syntax , 135
 - NULL
 - UltraLite ISNULL function, 273
 - NULL value in UltraLite
 - about, 207
 - NULL values
 - UltraLite SQL, 207
 - NULLIF function
 - UltraLite about, 288
 - using with CASE expressions in UltraLite, 218
 - numbers
 - UltraLite SQL, 206
 - numbers in UltraLite
 - about, 206
 - NUMERIC data type
 - UltraLite, 212
 - numeric functions
 - UltraLite alphabetical list, 239
 - numeric precision
 - UltraLite precision reference, 114
 - UltraLite precision usage, 36
- O**
- obfuscate database property
 - UltraLite reference, 110
 - UltraLite usage, 38
 - obfuscation
 - UltraLite usage, 38
 - on-device creation
 - about, 29
 - opening connections with connection strings
 - about, 47
 - operating systems
 - UltraLite, 12
 - UltraLite Windows supported platforms, 12
-

- operator precedence
 - UltraLite SQL syntax, 229
- operators
 - arithmetic operators for UltraLite, 228
 - bitwise operators for UltraLite, 229
 - comparison operators for UltraLite, 222
 - logical operators for UltraLite, 223
 - precedence of operators for UltraLite, 229
 - string operators for UltraLite, 229
 - UltraLite SQL syntax, 228
- operators in UltraLite
 - about, 228
- optimization
 - UltraLite queries, 343
 - UltraLite SQL, 232
- optimizer, vii
 - (see also query optimizer)
 - UltraLite impact of, 232
 - UltraLite overriding, 232
 - UltraLite plan interpretation, 233
 - UltraLite plan operations, 233
 - UltraLite using, 232
- options
 - UltraLite browsing, 78
 - UltraLite nearest_century reference, 109
 - UltraLite nearest_century usage, 35
 - UltraLite utilities, 186
- options (UltraLite)
 - global_id reference, 104
 - ml_remote_id reference, 108
- OR
 - bitwise operators for UltraLite, 229
 - logical operators for UltraLite, 223
- ORDER BY clause
 - UltraLite SELECT statement, 343
- order of operations
 - SQL operator precedence for UltraLite, 229
- outer references
 - subqueries for UltraLite SQL, 219
- overhead
 - UltraLite considering for reserve size, 148
- P**
- page_size database property
 - UltraLite reference, 112
 - UltraLite usage, 36
- pages
 - UltraLite size of, 36
- Palm Computing Platform, vii
 - (see also Palm OS)
- Palm data management utility
 - syntax, 185
- Palm HotSync Conduit installer utility
 - syntax, 163
- Palm OS
 - expansion cards, using, 51
 - UltraLite character sets, 32
 - UltraLite databases, 51
 - UltraLite PDB records, 51
 - UltraLite VFS databases, 51
- PALM_ALLOW_BACKUP connection parameter
 - UltraLite syntax, 145
- PALM_DB connection parameter
 - file database name for UltraLite, 139
- PALM_FILE connection parameter
 - UltraLite syntax , 137
- parameters
 - UltraLite connection list, 44
 - UltraLite connection overview, 47
 - UltraLite SQL input, 220
- partitioning
 - UltraLite rows publishing, 73
- Password connection parameter
 - UltraLite syntax, 146
- passwords
 - PWD UltraLite connection parameter, 146
 - UltraLite adding new, 45
 - UltraLite changing, 76
 - UltraLite considerations, 76
 - UltraLite databases, 45
 - UltraLite defaults, 76
 - UltraLite semantics, 46
- paths
 - UltraLite connection parameters, 50
- PATINDEX function
 - UltraLiteSQL syntax, 289
- pattern matching
 - UltraLitePATINDEX function, 289
 - wildcards, 289
- PDB, vii
 - (see also UltraLite databases and Palm OS)
 - UltraLite databases, 51
- performance
 - ulcond10 cache for UltraLite, 163
 - UltraLite CACHE_SIZE , 128

-
- UltraLite indexes, 27, 65, 69
 - UltraLite page sizes, 36
 - UltraLite preventing memory failures, 148
 - UltraLite query optimization, 343
 - persistent memory
 - UltraLite database storage, 50
 - physical limitations
 - UltraLite, 8
 - PI function
 - UltraLite SQL syntax, 290
 - placeholder
 - UltraLite SQL input parameter, 220
 - planning for scalability
 - UltraLite about, 10
 - plans
 - UltraLite cursors, 266
 - UltraLite operations for, 233
 - UltraLite plan interpretation, 233
 - UltraLite plan operations, 233
 - UltraLite queries, overriding, 232
 - UltraLite queries, reading, 233
 - UltraLite queries, working with, 232
 - UltraLite SQL syntax, 266
 - UltraLite text plans, 232
 - platforms
 - UltraLite file storage, 50
 - UltraLite multiple connection parameters for, 50
 - POWER function
 - UltraLite SQL syntax, 291
 - precedence
 - SQL operator precedence for UltraLite, 229
 - precision database property
 - UltraLite reference, 114
 - UltraLite usage, 36
 - predicates
 - ALL for UltraLite SQL, 224
 - ANY for UltraLite SQL, 225
 - BETWEEN for UltraLite SQL, 226
 - comparison operators for UltraLite, 222
 - EXISTS for UltraLite SQL, 226
 - IN for UltraLite SQL, 227
 - UltraLite SQL, 221
 - prefix
 - UltraLite for Windows CE databases , 51
 - prepared statements
 - UltraLite input parameters, 220
 - primary keys
 - generating unique values, 287
 - generating unique values using UUIDs, 287
 - UltraLite indexing, 27
 - UltraLite integrity constraints, 331
 - UltraLite order of columns, 331
 - UltraLite tables, 56
 - UltraLiteUUIDs and GUIDs, 287
 - procedures
 - UltraLite limitations, 5
 - programming interfaces
 - UltraLite supported, 13
 - properties
 - UltraLite browsing, 78
 - UltraLite databases, 30
 - UltraLite system table, 197
 - properties (UltraLite)
 - case reference, 94
 - checksum_level reference, 95
 - date_format reference, 97
 - date_order reference, 100
 - fips property reference, 102
 - nearest_century reference, 109
 - obfuscate reference, 110
 - page_size reference, 112
 - precision reference, 114
 - scale reference, 116
 - time_format reference, 118
 - timestamp_format reference, 120
 - timestamp_increment reference, 123
 - utf8_encoding reference, 125
 - publications
 - UltraLite altering , 326
 - UltraLite copying method, 63
 - UltraLite CREATE PUBLICATION statement, 330
 - UltraLite dropping , 75
 - UltraLite limit of, 172
 - UltraLite maximum of, 8
 - UltraLite publishing tables, 72
 - UltraLite rows publishing, 73
 - UltraLite SQL CREATE INDEX statement, 328
 - UltraLite SQL DROP INDEX statement, 337
 - UltraLite SQL DROP PUBLICATION statement, 338
 - UltraLite SQL DROP TABLE statement, 339
 - UltraLite system table, 196
 - UltraLite WHERE clause usage, 73
 - UltraLite working with, 72
 - publishing
-

- UltraLite rows, 73
- UltraLite tables, 72
- publishing a subset of rows from an UltraLite table
 - about, 73
- publishing whole UltraLite tables
 - about, 72
- PWD connection parameter
 - UltraLite syntax, 146

Q

- QUARTER function
 - UltraLite SQL syntax, 291
- queries
 - UltraLite optimization, 343
- query access plans in UltraLite
 - about, 232
- query optimization, vii
 - (see also optimizer)
 - UltraLite SQL, 232
- query optimizer, vii
 - (see also optimizer)
 - UltraLite, 232
- query plans
 - UltraLite how to read, 233
 - UltraLite operations, 233
 - UltraLite overriding, 232
 - UltraLite text of, 232
 - UltraLite working with, 232

R

- RADIANS function
 - UltraLite SQL syntax, 292
- range
 - UltraLite date type, 212
- reading
 - rows in UltraLite, 18
- reading UltraLite access plans
 - about, 233
- REAL data type
 - UltraLite, 212
- recovery
 - UltraLite databases, 18, 19
- reference databases
 - creating for UltraLite, 26
 - options for UltraLite, 27
- referential integrity
 - UltraLite databases, 5

- UltraLite indexes, 65
- registry keys
 - ulcond cache size, 163
- REMAINDER function
 - UltraLite SQL syntax, 292
- remote databases
 - deleting UltraLite data, 185
- remote ID considerations for MobiLink server synchronization
 - about, 40
- remote IDs
 - setting in UltraLite databases, 40
 - UltraLite about, 40
- remote servers
 - UltraLite creating tables , 331
- removing
 - UltraLite users, 77
- renaming
 - UltraLite columns, 322
 - UltraLite constraints , 322
 - UltraLite tables, 322
- REPEAT function
 - UltraLite SQL syntax, 293
- REPLACE function
 - UltraLite SQL syntax, 294
- REPLICATE function
 - UltraLite SQL syntax, 295
- requests
 - concurrency in UltraLite, 17
- RESERVE_SIZE connection parameter
 - UltraLite syntax, 148
- reserved words
 - UltraLite SQL, 202
- restoring
 - UltraLite databases, 19
- return codes
 - Interactive SQL [dbisql] utility for UltraLite, 157
- RIGHT function
 - UltraLite SQL syntax, 295
- RIGHT OUTER JOIN clause
 - UltraLite SQL syntax, 344
- role names
 - UltraLite foreign keys, 331
 - UltraLite role names, 331
- role of user authentication
 - UltraLite about, 45
- ROLLBACK statement
 - UltraLite SQL syntax, 341

- rollbacks
 - UltraLite databases, 18
- rolling back
 - UltraLite transaction overview , 19
 - UltraLite transactions, 341
- ROUND function
 - UltraLite SQL syntax, 296
- rounding
 - UltraLite scale reference, 116
 - UltraLite scale usage , 36
- row packing
 - UltraLite observing, 148
- rowlimit operation
 - UltraLite query access plans, 233
- rows
 - UltraLite deleting all rows from a table, 349
 - UltraLite inserting rows, 340
 - UltraLite publishing, 73
 - UltraLite selecting, 342
 - UltraLite updating rows, 352
- RTRIM function
 - UltraLiteSQL syntax, 297
- runtime
 - UltraLite files for, 11
- runtime library
 - UltraLite, 11

S

- sample application
 - starting CustDB in UltraLite, 83
- samples, vii
 - (see also tutorials)
- samples-dir
 - documentation usage, xii
- scale database property
 - UltraLite reference, 116
 - UltraLite usage, 36
- scan operation
 - UltraLite query access plans, 233
- schema
 - UltraLite catalog of tables for, 15
 - UltraLite changes, precautions, 15
- schemas
 - UltraLite system tables, 192
- scripted uploads
 - UltraLite CREATE PUBLICATION syntax, 330
- search conditions
 - ALL for UltraLite SQL, 224
 - ANY for UltraLite SQL, 225
 - BETWEEN for UltraLite SQL, 226
 - EXISTS for UltraLite SQL, 226
 - IN for UltraLite SQL, 227
 - UltraLite SQL, 221
- search conditions in UltraLite
 - about, 221
- SECOND function
 - UltraLite SQL syntax, 297
- SECONDS function
 - UltraLite SQL syntax, 298
- security
 - UltraLite, 38, 102
 - UltraLite user authentication, 45
- security considerations
 - UltraLite about, 38
- SELECT statement
 - UltraLite browsing data usage, 62
 - UltraLite object copying usage, 63
 - UltraLite SQL syntax, 342
- selecting
 - UltraLite forming unions, 351
 - UltraLite selecting rows, 342
- sensitivity
 - UltraLite case reference, 94
 - UltraLite case usage, 33
- SHORT_PLAN function
 - UltraLiteSQL syntax, 299
- SIGN function
 - UltraLite SQL syntax, 300
- SIMILAR function
 - UltraLite SQL syntax, 300
- SIN function
 - UltraLite SQL syntax, 301
- slash-asterisk
 - UltraLite comment indicator, 205
- SMALLINT data type
 - UltraLite, 212
- SMALLMONEY
 - UltraLite equivalent of, 214
- sort order
 - UltraLite collations, 31
- SOUNDEX function
 - UltraLite SQL syntax, 302
- SPACE function
 - UltraLite SQL syntax, 303
- special values

- UltraLite CURRENT DATE, 208
- UltraLite CURRENT TIME, 208
- UltraLite CURRENT TIMESTAMP, 209
- UltraLite SQL, 208
- UltraLite SQLCODE, 210
- SQL, vii
 - (see also UltraLite SQL)
 - comparison operators for UltraLite, 222
 - data types in UltraLite, 212
 - expressions in UltraLite, 215
 - operators in UltraLite, 228
 - search conditions in UltraLite, 221
 - UltraLite identifiers , 203
 - UltraLite keywords , 202
 - UltraLite numbers, 206
 - UltraLite reserved words, 202
 - UltraLite schema changes with, 15
 - UltraLite statement types, 320
 - UltraLite strings, 204
 - variables in UltraLite, 231
- SQL Anywhere
 - documentation, viii
 - statistics, comparing with UltraLite, 8
- SQL Anywhere databases
 - database comparison, 5
- SQL functions
 - types of function for UltraLite, 236
 - UltraLite aggregate, 236
 - UltraLite data type conversion, 236
 - UltraLite date and time, 236
 - UltraLite miscellaneous, 238
 - UltraLite numeric, 239
 - UltraLite string, 239
- SQL preprocessor utility
 - UltraLite syntax , 158
- SQL statements
 - UltraLite COMMIT syntax, 327
 - UltraLite CREATE PUBLICATION syntax, 330
 - UltraLite CREATE TABLE syntax , 331
 - UltraLite DELETE dynamic SQL syntax, 336
 - UltraLite FROM clause syntax, 344
 - UltraLite INSERT syntax, 340
 - UltraLite SQL ALTER PUBLICATION syntax, 326
 - UltraLite SQL ALTER TABLE syntax, 322
 - UltraLite SQL CREATE INDEX syntax, 328
 - UltraLite SQL DROP INDEX syntax, 337
 - UltraLite SQL DROP PUBLICATION syntax, 338
 - UltraLite SQL DROP TABLE syntax, 339
 - UltraLite SQL ROLLBACK syntax, 341
 - UltraLite SQL SELECT syntax, 342
 - UltraLite SQL STOP SYNCHRONIZATION DELETE syntax, 348
 - UltraLite SQL UNION syntax, 351
 - UltraLite START SYNCHRONIZATION DELETE syntax, 347
 - UPDATE syntax for UltraLite SQL, 352
- SQL syntax
 - alphabetical list of all functions, 241
 - CASE expression for UltraLite, 217
 - IF expressions for UltraLite, 217
 - UltraLite column names, 216
 - UltraLite comments, 205
 - UltraLite constants, 216
 - UltraLite functions, 236
 - UltraLite input parameters, 220
 - UltraLite special values, 208
 - UltraLite SQLCODE special value, 210
- SQLCODE
 - UltraLite concurrency checks, 17
 - UltraLite special value, 210
- SQLCODE SQLE_LOCKED
 - UltraLite concurrency error, 17
- SQLE_NOTFOUND
 - UltraLite concurrency error, 17
- sqlpp utility
 - syntax for UltraLite, 158
- SQRT function
 - UltraLite SQL syntax, 303
- square root function
 - UltraLite SQRT function, 303
- START connection parameter
 - UltraLite syntax, 149
- START SYNCHRONIZATION DELETE statement
 - UltraLite SQL syntax, 347
- state bytes
 - UltraLite databases, 18
- statement syntax
 - UltraLite FROM clause, 344
- statements
 - UltraLite COMMIT syntax , 327
 - UltraLite CREATE PUBLICATION syntax, 330
 - UltraLite CREATE TABLE syntax , 331
 - UltraLite DELETE dynamic SQL syntax, 336
 - UltraLite FROM clause, 344
 - UltraLite INSERT syntax, 340

- UltraLite prepared, input parameters for, 220
- UltraLite SQL ALTER PUBLICATION syntax, 326
- UltraLite SQL ALTER TABLE syntax, 322
- UltraLite SQL CREATE INDEX syntax, 328
- UltraLite SQL DROP INDEX syntax, 337
- UltraLite SQL DROP PUBLICATION syntax, 338
- UltraLite SQL DROP TABLE syntax, 339
- UltraLite SQL ROLLBACK syntax, 341
- UltraLite SQL SELECT syntax, 342
- UltraLite SQL STOP SYNCHRONIZATION DELETE syntax, 348
- UltraLite SQL UNION syntax, 351
- UltraLite START SYNCHRONIZATION DELETE syntax, 347
- UltraLite TRUNCATE TABLE syntax, 349
- UltraLite types, 320
- UPDATE syntax for UltraLite SQL, 352
- statistics
 - UltraLite, 8
- STOP SYNCHRONIZATION DELETE statement
 - UltraLite SQL syntax, 348
- storage
 - PALM_ALLOW_BACKUP for UltraLite, 145
 - UltraLite reserve size, 148
- stored procedures
 - UltraLite limitations, 5
- STR function
 - UltraLite SQL syntax, 304
- STRING function
 - UltraLite SQL syntax, 305
- string functions
 - UltraLite alphabetical list, 239
- string length
 - UltraLite LENGTH function, 275
- string literals
 - UltraLite about, 216
- string operators
 - dynamic SQL syntax for UltraLite, 229
- string position
 - UltraLite LOCATION function, 278
- strings
 - removing trailing blanks in UltraLite, 297
 - replacing in UltraLite , 294
 - UltraLite ambiguous conversions to dates, 35, 109
 - UltraLite case sensitivity, 204
 - UltraLite SQL, 204
 - UltraLite SQL functions, 239
 - strings in UltraLite
 - about, 204
 - strong encryption
 - UltraLite reference, 102
 - UltraLite usage, 38
 - STRTOUUID function
 - UltraLiteSQL syntax, 305
 - STUFF function
 - UltraLite SQL syntax, 306
 - subqueries
 - UltraLite SQL, 219
 - subquery operation
 - UltraLite query access plans, 233
 - SUBSTR function
 - UltraLite SQL syntax, 307
 - SUBSTRING function
 - UltraLite SQL syntax, 307
 - substrings
 - replacing in UltraLite , 294
 - UltraLite about, 307
 - SUM function
 - UltraLite SQL syntax, 308
 - supplying UltraLite connection parameters
 - about, 47
 - support
 - newsgroups, xv
 - Sybase Central
 - browsing CustDB in UltraLite, 88
 - creating UltraLite databases, 25
 - UltraLite column creation methods, 58
 - UltraLite copying database objects method, 63
 - UltraLite creating indexes, 67
 - UltraLite creating table methods, 56
 - UltraLite system table browsing methods, 61
 - UltraLite table alteration methods, 59
 - UltraLite table browsing methods, 61
 - UltraLite table creation methods, 56
 - UltraLite table deletion methods, 60
 - Symbian
 - UltraLite character sets, 32
 - SYMBIAN_FILE connection parameter
 - file database name for UltraLite, 140
 - synchronization
 - character sets in UltraLite, 31
 - setting timestamp_increment reference, 123
 - ulsync utility for UltraLite databases, 178
 - UltraLite CustDB tutorial, 86
 - UltraLite schema changes during, 15

- UltraLite system table, 192
- synchronization logic
 - browsing Sybase Central in UltraLite, 88
- synchronization model
 - UltraLite databases, 26
- synchronization parameters
 - UltraLite Disable Concurrency overview, 18
- synchronization scripts
 - browsing the UltraLite sample, 88
- syntax
 - arithmetic operators for UltraLite, 228
 - bitwise operators for UltraLite, 229
 - CASE expression for UltraLite, 217
 - comparison operators for UltraLite, 222
 - IF expressions for UltraLite, 217
 - logical operators for UltraLite, 223
 - SQL operator precedence for UltraLite, 229
 - string operators for UltraLite, 229
 - UltraLite column names, 216
 - UltraLite constants, 216
 - UltraLite CURRENT DATE special value, 208
 - UltraLite CURRENT TIMESTAMP special value, 209
 - UltraLite special values, 208
 - UltraLite SQL comments, 205
 - UltraLite SQL CURRENT TIME special value, 208
 - UltraLite SQL functions, 236
 - UltraLite SQL input parameters, 220
 - UltraLite SQL operators, 228
 - UltraLite SQLCODE special value, 210
- SYS
 - UltraLite system tables, 192
- sysarticle system table [UltraLite]
 - about, 196
- syscolumn system table [UltraLite]
 - about, 193
- sysindex system table [UltraLite]
 - about, 194
- sysixcol system table [UltraLite]
 - about, 195
- syspublication system table [UltraLite]
 - about, 196
- systable system table [UltraLite]
 - about, 192
- system failures
 - UltraLite databases, 19
- system functions

- UltraLite limitations, 5
- system objects
 - UltraLite displaying methods, 62
- system tables
 - UltraLite about, 192
 - UltraLite browsing methods, 61
 - UltraLite hiding and showing, 192
 - UltraLite sysarticle, 196
 - UltraLite syscolumn, 193
 - UltraLite sysindex, 194
 - UltraLite sysixcol, 195
 - UltraLite syspublication, 196
 - UltraLite systable, 192
 - UltraLite sysuldata, 197
- sysuldata system table [UltraLite]
 - about, 197

T

- table constraints
 - UltraLite adding, deleting, or modifying, 322
 - UltraLite CREATE TABLE statement, 331
- table expressions
 - subqueries for UltraLite SQL, 219
- table owners
 - UltraLite, 203
- tableOrder
 - UltraLite ulsync options for, 178
- tables
 - UltraLite altering, 322
 - UltraLite altering methods, 59
 - UltraLite browsing methods, 61
 - UltraLite copying method, 63
 - UltraLite copying methods, 62
 - UltraLite creating , 331
 - UltraLite creating methods, 56
 - UltraLite deleting methods, 60
 - UltraLite editing methods, 61
 - UltraLite inserting rows, 340
 - UltraLite limitations, 8
 - UltraLite renaming, 322
 - UltraLite replicating , 322
 - UltraLite rows publishing, 73
 - UltraLite table filtering methods, 62
 - UltraLite temporary, 16, 232
 - UltraLite truncating, 349
 - UltraLite working with, 56
- TAN function

-
- UltraLite SQL syntax, 309
 - target platforms
 - UltraLite, 12
 - TCP/IP, vii
 - (see also TCP/IP synchronization)
 - technical support
 - newsgroups, xv
 - temp operation
 - UltraLite query access plans, 233
 - temporary files
 - UltraLite, 16
 - temporary tables
 - UltraLite about, 16
 - UltraLite limitations, 5
 - UltraLite queries, 232
 - text
 - UltraLite equivalent of, 214
 - THEN
 - IF expressions for UltraLite, 217
 - threads
 - concurrency in UltraLite, 17
 - time considerations
 - UltraLite about, 34
 - TIME data type
 - UltraLite , 212
 - time functions
 - UltraLite alphabetical list, 236
 - time_format database property
 - UltraLite reference, 118
 - UltraLite usage, 34
 - times
 - UltraLite conversion functions, 236
 - TIMESTAMP
 - UltraLite column limitations, 5
 - UltraLite TIMESTAMP columns, 331
 - timestamp considerations
 - UltraLite about, 35
 - TIMESTAMP data type
 - UltraLite, 212
 - timestamp_format database property
 - UltraLite reference, 120
 - UltraLite usage, 35
 - timestamp_increment database property
 - UltraLite reference, 123
 - UltraLite usage, 35
 - timestamp_increment property
 - using in MobiLink synchronization reference, 123
 - TINYINT data type
 - UltraLite, 212
 - TODAY function
 - UltraLite SQL syntax, 310
 - TOP clause
 - UltraLite SQL SELECT statement, 342
 - transaction log
 - UltraLite internal, 15, 19
 - transactions
 - concurrency in UltraLite, 17
 - UltraLite committing, 327
 - UltraLite databases, 18
 - UltraLite rolling back, 341
 - transport-layer security, vii
 - (see also TLS)
 - triggers
 - UltraLite limitations, 5
 - TRIM function
 - UltraLite SQL syntax, 310
 - troubleshooting
 - UltraLite connections, 47, 49
 - TRUNCATE TABLE statement
 - UltraLite SQL syntax, 349
 - truncating
 - UltraLite tables, 349
 - TRUNCNUM function
 - UltraLite SQL syntax, 311
 - tuning
 - UltraLite indexes, 33
 - tuning performance with index hashing
 - about, 69
 - tutorials
 - UltraLite CustDB database synchronization, 86
 - UltraLite CustDB files, 81
 - UltraLite CustDB introduction, 80
 - UltraLite CustDB sample, 79
- ## U
- UCASE function
 - UltraLite SQL syntax, 311
 - UDB, vii
 - (see also UltraLite databases)
 - UID connection parameter
 - UltraLite syntax, 150
 - ulafreg utility
 - syntax, 161
 - ulcond10 utility
 - syntax, 163

- ulcreate utility
 - syntax, 165
 - using, 25
- ULDBUtil
 - about, 185
- uleng10 utility
 - syntax, 168
- ulinfo utility
 - UltraLite syntax, 169
- ulinit utility
 - syntax, 172
 - using, 26
- ulload utility
 - syntax, 174
 - UltraLite enhancement in version 10.0.0, 178
 - using, 28
- ULSQLCONNECT environment variable
 - about, 47
 - description, 53
- ulstop utility
 - syntax, 177
- ulsync utility
 - extended options, 187
 - syntax, 178
- UltraLite, vii
 - (see also UltraLite databases and UltraLite applications)
 - about, 3
 - API choosing , 13
 - comma-separated lists, 277
 - data conversion, 236
 - data management options, 11
 - deployment options, 11
 - development platforms, 12
 - embedded engine client files, 11
 - error codes, 189
 - libraries choosing , 13
 - multi-threaded applications, 11
 - runtime files, 11
 - SQL functions, aggregate, 236
 - SQL functions, data type conversion, 236
 - SQL functions, date and time, 236
 - SQL functions, miscellaneous, 238
 - SQL functions, numeric, 239
 - SQL functions, string, 239
 - SQL functions, types of, 236
 - SQL statement reference, 319
 - statistics, comparing with SQL Anywhere, 8
 - supported APIs, 12
 - supported Windows platforms, 12
 - table owners, 203
 - utility programs, 153, 154
- UltraLite APIs (see UltraLite C/C++ API) (see UltraLite for M-Business Anywhere API) (see UltraLite AppForge API) (see UltraLite.NET API)
- UltraLite applications, vii
 - (see also UltraLite for M-Business Anywhere API)
 - (see also UltraLite.NET API)
 - (see also UltraLite AppForge API)
 - (see also UltraLite C/C++ API)
- UltraLite connection parameters
 - about, 47
 - list of, 44
- UltraLite connection string parameters reference
 - about, 127
- UltraLite data and state management
 - about, 16
- UltraLite database creation utility
 - syntax, 165
- UltraLite database settings reference
 - about, 93
- UltraLite database synchronization utility
 - extended options, 187
- UltraLite databases
 - about, 15
 - backing up on Windows CE, 51
 - collation sequences, 31
 - columns adding, 58
 - columns altering, 59
 - concurrency, 17
 - connection overview, 44
 - connection parameter list, 44
 - connection parameters overview, 47
 - create database wizard, using, 25
 - creating, 24
 - creating from SQL Anywhere reference database, 26
 - creating from the command prompt, 25
 - database comparison, 5
 - deleting application data from Palm OS device, 185
 - desktop creation options, 24
 - encrypting, 38, 102
 - entity-relationship diagrams, 63
 - environment variables, 49
 - features, 4

- file internals, 15
- file storage, 15, 50
- file storage on Palm OS, 51
- indexes creating, 328
- indexes hashing overview, 33
- indexes hashing reference, 106
- indexes types of, 66
- indexes working with, 65
- indexes, when to create, 67
- indexing primary keys, 27
- initializing from Sybase Central, 25
- initializing from the command prompt, 26
- introduction, 4, 15
- limitations, 8
- managing multiple, 17
- methods of creating, 24
- MobiLink synchronization, 192
- modeling from MobiLink, 26
- objects copy method, 63
- options browsing, 78
- page_size reference, 112
- page_size usage, 36
- Palm OS support, 51
- properties, 30
- properties browsing, 78
- publications about, 72
- publications dropping, 75
- publishing rows, 73
- publishing tables, 72
- schema, 192
- schema changes, 15
- schema overview, 15
- sourcing from XML, 28
- storage, 50
- system table indices, 194, 195
- system table properties, 197
- system table publications, 196
- system tables displaying, 192
- table synchronization suffixes, 57
- tables copying, 62
- tables creating, 56
- tables dropping, 60
- tables filtering, 62
- tables, browsing, 61
- temporary files, 16
- ULSQLCONNECT, 49
- unique keys, 66
- upgrade previous versions, 24
- user IDs, 45
- users adding, 76
- users deleting, 77
- UTF8BIN collation for UNICODE characters, 31
- Windows CE, 51
- Windows desktop, 51
- working with, 55
- UltraLite engine
 - about, 11
- UltraLite engine start utility
 - syntax, 168
- UltraLite engine stop utility
 - syntax, 177
- UltraLite information utility
 - about, 169
- UltraLite initialize database utility
 - about, 172
- UltraLite load XML to database utility
 - syntax, 174
- UltraLite passwords
 - about, 45
- UltraLite registry update utility
 - syntax, 161
- UltraLite runtime
 - about, 11
- UltraLite SQL
 - comments, 205
 - data types, 212
 - dates, 211
 - expressions, 215
 - identifiers, 203
 - keywords, 202
 - NULL values, 207
 - numbers, 206
 - operators, 228
 - query access plans for, 232
 - special values, 208
 - strings, 204
 - times, 211
 - variables, 231
- UltraLite SQL elements reference
 - about, 201
- UltraLite SQL statements
 - ALTER PUBLICATION statement syntax, 326
 - ALTER TABLE statement syntax, 322
 - categories of, 320
 - COMMIT statement syntax, 327
 - CREATE INDEX statement syntax, 328

- CREATE PUBLICATION statement syntax, 330
- CREATE TABLE statement syntax, 331
- DELETE statement syntax, 336
- DROP INDEX statement syntax, 337
- FROM clause, 344
- INSERT statement, 340
- overview, 319
- ROLLBACK statement syntax, 341
- SELECT statement syntax, 342
- START SYNCHRONIZATION DELETE statement syntax, 347
- STOP SYNCHRONIZATION DELETE statement syntax, 348
- TRUNCATE TABLE syntax, 349
- UNION operation syntax, 351
- UPDATE statement syntax, 352
- UltraLite SQL statements overview
 - about, 320
- UltraLite synchronization
 - remote IDs and user IDs, 40
- UltraLite synchronization utility
 - syntax, 178
- UltraLite system table reference
 - about, 191
- UltraLite system tables
 - about, 192
- UltraLite temporary files
 - about, 16
- UltraLite unload database utility
 - syntax, 180
- UltraLite unload old database utility
 - syntax, 183
- UltraLite user IDs
 - about, 45
 - MobiLink uniqueness, 40
- UltraLite utilities reference
 - about, 153
- UltraLite-specific decisions you need to make
 - about, 10
- UltraLite.NET
 - UltraLite engine support, 11
- ulunload utility
 - syntax, 180
- ulunloadold utility
 - syntax, 183
- understanding database management fundamentals for UltraLite
 - about, 15
- undoing
 - UltraLite transactions, 341
- UNICODE characters
 - UltraLite collation for, 31
- UNION operation
 - UltraLite SQL syntax, 351
- union-all operation
 - UltraLite query access plans, 233
- unions
 - UltraLite multiple select statements, 351
- unique constraints
 - UltraLite copying method, 63
 - UltraLite CREATE TABLE statement, 331
- unique indexes
 - UltraLite databases, 328
 - UltraLite index creation from, 66
- unique keys
 - UltraLite index creation from, 66
- UNIQUEIDENTIFIER data type
 - UltraLite, 212
- universally unique identifiers, vii
 - (see also UUIDs)
 - UltraLiteSQL syntax for NEWID function, 287
- unload old database utility [ulunloadold]
 - syntax, 183
- unloading
 - UltraLite databases, 180
 - UltraLite databases from earlier versions, 183
- unnamed foreign keys
 - UltraLite usage, 331
- UPDATE statement
 - UltraLite SQL syntax, 352
- updates
 - UltraLite databases, 18
- updating
 - UltraLite updating rows, 352
- UPPER function
 - UltraLite SQL syntax, 312
- uppercase characters
 - UPPER function, 312
- uppercase strings
 - UltraLite UCASE function, 311
 - UltraLite UPPER function, 312
- user authentication
 - PWD UltraLite connection parameter, 146
 - UltraLite, 45
 - UltraLite bypassing, 45
 - UltraLite databases, 45

-
- UltraLite setup, 45
 - user IDs
 - UltraLite adding new, 45
 - UltraLite changing, 76
 - UltraLite considerations, 76
 - UltraLite databases, 45
 - UltraLite defaults, 76
 - UltraLite semantics, 46
 - user-defined data types
 - UltraLite equivalents of, 214
 - unsupported in UltraLite, 212
 - users
 - UltraLite adding, 76
 - UltraLite deleting, 77
 - UltraLite working with, 76
 - using allsync and nosync suffixes
 - UltraLite about, 57
 - utf8_encoding database property
 - UltraLite reference, 125
 - UltraLite usage, 32
 - UTF8BIN collation
 - UltraLite considerations, 31
 - utilities
 - Interactive SQL [dbisql] syntax for UltraLite, 155
 - ulcreate, 165
 - ulload, 174
 - UltraLite AppForge registration [ulafreg] utility, 161
 - UltraLite database creation [ulcreate] utility, 165
 - UltraLite engine start [uleng10] utility, 168
 - UltraLite engine stop [ulstop] utility, 177
 - UltraLite error codes, 189
 - UltraLite HotSync Conduit installer [ulcond10] utility, 163
 - UltraLite information [ulinfo] utility, 169
 - UltraLite initialize database [ulinit] utility, 172
 - UltraLite load XML to database [ulload] utility, 174
 - UltraLite Palm [ULDBUtil] utility, 185
 - UltraLite SQL Preprocessor [sqlpp] utility, 158
 - UltraLite synchronization [ulsync] , 178
 - UltraLite unload data to XML [ulunload] utility, 180
 - UltraLite unload old database [ulunloadold] utility, 183
 - UUIDs
 - UltraLite SQL syntax for UUIDTOSTR function, 313
 - UltraLiteSQL syntax for NEWID function, 287
 - UltraLiteSQL syntax for STRTOUUID function, 305
 - UUIDTOSTR function
 - UltraLite SQL syntax, 313
- ## V
- validating
 - UltraLite checksum usage, 37
 - UltraLite checksum_level reference, 95
 - values
 - UltraLite index hash, 33
 - VARBINARY data type
 - UltraLite, 212
 - VARCHAR data type
 - UltraLite, 212
 - variables
 - UltraLite SQL, 231
 - variables in UltraLite
 - about, 231
 - VFS
 - UltraLite databases, 51
 - viewing
 - UltraLite table methods, 61
 - viewing an UltraLite query access plan
 - about, 232
 - viewing entity-relationship diagrams from the UltraLite plug-in
 - about, 63
 - viewing UltraLite database settings
 - about, 78
 - virtual file system (see VFS)
 - Visual Basic compatibility
 - UltraLite support, 13
- ## W
- WEEKS function
 - UltraLite SQL syntax, 314
 - WHEN
 - CASE expression for UltraLite, 217
 - when to use an index
 - UltraLite about, 65
 - when to view a query access plan
 - about, 232
 - WHERE clause
 - UltraLite ALTER PUBLICATION statement, 326
-

- UltraLite CREATE PUBLICATION statement, 330
- UltraLite DELETE statement, 336
- UltraLite publication usage, 73
- UltraLite SELECT statement, 342
- UltraLite UPDATE statement, 352
- wildcards
 - pattern matching, 289
- Windows
 - UltraLite character sets, 32
- windows, vii
 - (see also Windows ME)
 - (see also Windows Mobile 5)
 - (see also Windows NT)
 - (see also Windows Vista)
 - (see also Windows XP/200x)
- Windows CE
 - UltraLite character sets, 32
 - UltraLite database prefix, 51
 - UltraLite databases, 51
 - UltraLite engine support, 11
 - UltraLite FIPS enablement, 102
- Windows desktop
 - UltraLite databases, 51
 - UltraLite engine support, 11
- wizards
 - UltraLite create database, 25
 - UltraLite index creation, 67
 - UltraLite publication creation tables, 72
- words
 - UltraLite keywords, 202
 - UltraLite reserved words, 202
- working with indexes
 - UltraLite about, 65
- working with UltraLite databases
 - about, 55
- working with UltraLite indexes
 - about, 65
- working with UltraLite publications
 - about, 72
- working with UltraLite tables and columns
 - about, 56
- working with UltraLite users
 - about, 76

X

XML

- loading to database, 174
- sourcing UltraLite databases from , 28
- UltraLite equivalent of, 214
- unloading database to, 178

Y

- YEAR function
 - UltraLite SQL syntax, 315
- YEARS function
 - UltraLite SQL syntax, 315
- YMD function
 - UltraLite SQL syntax, 316

Z

- zero-padding
 - UltraLite date_format reference, 98, 121