MobiLink
Getting Started

June 2008

Version 11.0.0

Copyright and trademarks

Contents

About this book

Subject

This book introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.

Audience

This book is for users of SQL Anywhere and other relational database systems who want to add synchronization to their information systems.

Before you begin

For a comparison of MobiLink with other synchronization and replication technologies, see "Overview of data exchange technologies" [*SQL Anywhere 11 - Introduction*].

About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in three formats that contain identical information.

- **HTML Help** The online Help contains the complete SQL Anywhere documentation, including the books and the context-sensitive help for SQL Anywhere tools. The online Help is updated with each release of the product.

  If you are using a Microsoft Windows operating system, the online Help is provided in HTML Help (CHM) format. To access the documentation, choose **Start** » **Programs** » **SQL Anywhere 11** » **Documentation** » **Online Books**.

  The administration tools use the same online documentation for their Help features.

- **Eclipse** On Unix platforms, the complete online Help is provided in Eclipse format. To access the documentation, run *sadoc* from the *bin32* or *bin64* directory of your SQL Anywhere 11 installation.

- **PDF** The complete set of SQL Anywhere books is provided as a set of Portable Document Format (PDF) files. You must have a PDF reader to view information. To download Adobe Reader, visit Adobe.com.

  To access the PDF documentation on Microsoft Windows operating systems, choose **Start** » **Programs** » **SQL Anywhere 11** » **Documentation** » **Online Books - PDF Format**.

  To access the PDF documentation on Unix operating systems, use a web browser to open *install-dir/documentation/en/pdf/index.html*.

About the books in the documentation set

SQL Anywhere documentation

The SQL Anywhere documentation consists of the following books:

- **SQL Anywhere 11 - Introduction** This book introduces SQL Anywhere 11—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 11 - Changes and Upgrading** This book describes new features in SQL Anywhere 11 and in previous versions of the software.
- **SQL Anywhere Server - Database Administration** This book describes how to run, manage, and configure SQL Anywhere databases. It describes database connections, the database server, database files, backup procedures, security, high availability, and replication with Replication Server, as well as administration utilities and options.
- **SQL Anywhere Server - Programming** This book describes how to build and deploy database applications using the C, C++, Java, PHP, Perl, Python, and .NET programming languages such as Visual Basic and Visual C#. A variety of programming interfaces such as ADO.NET and ODBC are described.
- **SQL Anywhere Server - SQL Reference** This book provides reference information for system procedures, and the catalog (system tables and views). It also provides an explanation of the SQL Anywhere implementation of the SQL language (search conditions, syntax, data types, and functions).

- **SQL 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.
- **MobiLink - Getting Started** This book introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.
- **MobiLink - Client Administration** This book describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases. This book also describes the Dbmlsync API, which allows you to integrate synchronization seamlessly into your C++ or .NET client applications.
- **MobiLink - Server Administration** This book describes how to set up and administer MobiLink applications.
- **MobiLink - Server-Initiated Synchronization** This book 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 book describes QAnywhere, which is 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.
- **UltraLite - Database Management and Reference** This book introduces the UltraLite database system for small devices.
- **UltraLite - C and C++ Programming** This book describes UltraLite C and C++ programming interfaces. With UltraLite, you can develop and deploy database applications to handheld, mobile, or embedded devices.
- **UltraLite - M-Business Anywhere Programming** This book 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 Mobile, or Windows.
- **UltraLite - .NET Programming** This book describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.
- **UltraLiteJ** This book describes UltraLiteJ. With UltraLiteJ, you can develop and deploy database applications in environments that support Java. UltraLiteJ supports BlackBerry smartphones and Java SE environments. UltraLiteJ is based on the iAnywhere UltraLite database product.
- **Error Messages** This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.

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.

Operating system conventions

SQL Anywhere runs on a variety of platforms, including Windows, Windows Mobile, Unix, Linux, and Mac OS X. To simplify references to operating systems, the documentation groups the supported operating systems as follows:

- **Windows** The Microsoft Windows family of operating systems for desktop and laptop computers. The Windows family includes Windows Vista and Windows XP.
- **Windows Mobile** Windows Mobile provides a Windows user interface and additional functionality, such as small versions of applications like Word and Excel. Windows Mobile is most commonly used on mobile devices.

Limitations or variations in SQL Anywhere are commonly based on the underlying operating system, and seldom on the particular variant used (Windows Mobile).

- **Unix** Unless specified, Unix refers to Linux, Mac OS X, and Unix platforms.

File name conventions

In many cases, references to file and directory names are similar on all supported platforms, with simple transformations between the various forms. To simplify the documentation in these cases, Windows conventions are used. In other cases, where the details are more complex, the documentation shows all relevant forms.

Here are the conventions used to simplify the documentation of file and directory names:

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

  After installation is complete, the environment variable SQLANY11 specifies the location of the installation directory containing the SQL Anywhere components (*install-dir*).

  For example, the documentation may refer to a file as *install-dir\readme.txt*. On Windows platforms, this reference is equivalent to *%SQLANY11%\readme.txt*. On Unix platforms, this reference is equivalent to *$SQLANY11/readme.txt*.

  For more information about the default location of *install-dir*, see "SQLANY11 environment variable" [*SQL Anywhere Server - Database Administration*].

- **Uppercase and lowercase directory names** On Windows, directory names often use mixed case. References to directory names can use any case, since the Windows file system is not case sensitive.

  Unix file systems are case sensitive. The use of mixed-case is less common.

  The SQL Anywhere installation program follows operating system conventions for its directory structure. On Windows, the installation contains directories such as *Bin32* and *Documentation*. On Unix, these directories are called *bin32* and *documentation*.

  The documentation often uses the mixed case forms of directory names. Usually, you can convert a mixed case directory name to lowercase for the equivalent directory name on Unix platforms. For example, the directory *MobiLink* is *mobilink* on Unix platforms.

- **Slashes separating parts of directory names** The documentation uses backslashes as the directory separator. For example, the PDF form of the documentation is found in the directory *install-dir\Documentation\en\pdf*, which is the Windows form.

  On Unix platforms, replace the backslash with the forward slash. The PDF documentation is found in the directory *install-dir/Documentation/en/pdf*.

- **Executable files** The documentation shows executable file names using Windows conventions, with a suffix such as *.exe* or *.bat*. On Unix platforms, executable file names have no suffix.

  For example, on Windows, the network database server is *dbsrv11.exe*. On Unix, Linux, and Mac OS X, it is *dbsrv11*.

- **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 SQLANYSAMP11 specifies the location of the directory containing the samples (*samples-dir*). From the Windows **Start** menu, choosing **Programs** » **SQL Anywhere 11** » **Sample Applications And Projects** opens a Windows Explorer window in this directory.

  For more information about the default location of *samples-dir*, by operating system, see "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*.

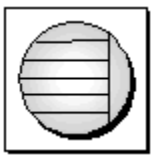
Graphic icons

The following icons are used in this documentation.

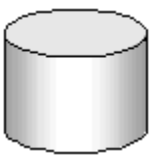
- A client application.

- A database server, such as Sybase SQL Anywhere.

- 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 programming interface.

Contacting the documentation team

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

To submit your comments and suggestions, send an email to the SQL Anywhere documentation team at iasdoc@sybase.com. Although we do not reply to emails, your feedback helps us to improve our documentation, so your input is welcome.

Finding out more and requesting technical support

Additional information and resources, including a code exchange, are available at the Sybase iAnywhere Developer Community at http://www.sybase.com/developer/library/sql-anywhere-techcorner.

If you have questions or need help, you can post messages to the Sybase iAnywhere 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 running the following command: **dbeng11 -v**.

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 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.

Part I. Using MobiLink Technology

This part introduces MobiLink synchronization technology and describes how to use it to replicate data between two or more databases.

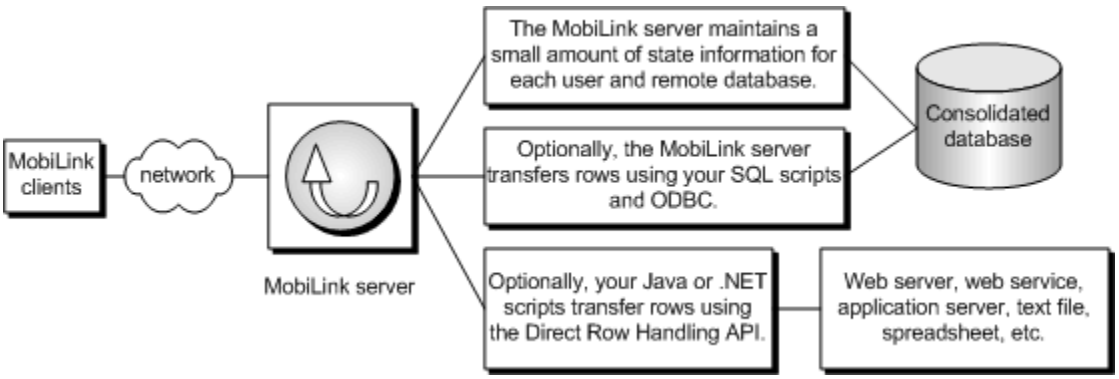
CHAPTER 1

Introducing MobiLink synchronization

Contents

Parts of a MobiLink application

In MobiLink synchronization, many clients synchronize through the MobiLink server to central data sources.



- **MobiLink clients** The client can be installed on a handheld device such as a Palm Pilot or Windows Mobile device, a server or desktop computer, or a smartphone. Two types of clients are supported: UltraLite and SQL Anywhere databases. Either or both can be used in a MobiLink installation. See "Introducing MobiLink clients" [*MobiLink - Client Administration*].
- **Network** The connection between the MobiLink server and the MobiLink client can use a number of protocols. See:
  - MobiLink server: "-x option" [*MobiLink - Server Administration*]
  - UltraLite and SQL Anywhere clients: "MobiLink client network protocol options" [*MobiLink - Client Administration*]
- **MobiLink server** This server manages the synchronization process and provides the interface between all MobiLink clients and the consolidated database server. See "MobiLink server" [*MobiLink - Server Administration*].
- **Consolidated database** This database typically holds system tables and procedures that are required by MobiLink synchronization, as well as state information needed to synchronize. It also typically contains the central copy of information in the synchronization system. See "MobiLink consolidated databases" [*MobiLink - Server Administration*].
- **State information** The MobiLink server typically maintains synchronization information in system tables in the consolidated database. It does this over an ODBC connection.

  You can also choose to store your state information in a separate database. See "MobiLink system database" [*MobiLink - Server Administration*].
- **SQL row handling** If you provide the MobiLink server with SQL scripts, it uses these scripts to transfer rows to and from the consolidated database over an ODBC connection. See "Options for writing server-side synchronization logic" on page 14.
- **Direct row handling** In addition to a consolidated database, you can optionally synchronize with other data sources using MobiLink direct row handling. See "Direct row handling" [*MobiLink - Server Administration*].

You write **synchronization scripts** for each table in the remote database and you save these scripts in MobiLink system tables in the consolidated database. These scripts determine what is done with the uploaded data, and what data to download. There are two types of script: table scripts and connection-level scripts. See:

- "Overview of MobiLink events" [*MobiLink - Server Administration*]
- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "Synchronization events" [*MobiLink - Server Administration*]
- "Options for writing server-side synchronization logic" on page 14

MobiLink features

MobiLink synchronization is adaptable and flexible. Following are some of its key features:

Features

- **Easy to get started** Using the **Create Synchronization Model Wizard**, you can create synchronization applications quickly. The wizard can handle many difficult implementation details of complex synchronization systems. Sybase Central Model mode allows you to view a synchronization model offline, provides an easy interface for making changes, and has a deployment option for you to deploy the model to your consolidated database.
- **Monitoring and reporting** MobiLink provides two mechanisms for monitoring your synchronizations: the MobiLink Monitor and statistical scripts.
- **Performance tuning** There are a number of mechanisms for tuning MobiLink performance. For example, you can adjust the degree of contention, upload cache size, number of database connections, logging verbosity, or BLOB cache size.
- **Scalability** MobiLink is an extremely scalable and robust synchronization platform. A single MobiLink server can handle thousands of simultaneous synchronizations, and multiple MobiLink servers can be run simultaneously using load balancing. The MobiLink server is multi-threaded and uses connection pooling with the consolidated database.
- **Security** MobiLink provides extensive security options, including user authentication that can be integrated with your existing authentication, encryption, and transport-layer security that works by the exchange of secure certificates. MobiLink also provides FIPS-certified security options.

Architecture

- **Data coordination** MobiLink allows you to choose selected portions of the data for synchronization. MobiLink synchronization also allows you to resolve conflicts between changes made in different databases. The synchronization process is controlled by synchronization logic, which can be written as a SQL, Java, or .NET application. Each piece of logic is called a **script**. With scripts, for example, you can specify how uploaded data is applied to the consolidated database, specify what gets downloaded, and handle different schema and names between the consolidated and remote databases. Event-based scripting provides great flexibility in the design of the synchronization process, including such features as conflict resolution, error reporting, and user authentication.
- **Two-way synchronization** Changes to a database can be made at any location.
- **Upload-only or download-only synchronization** You can choose to perform only an upload or only a download, as well as doing a two-way synchronization.
- **File-based download** Downloads can be distributed as files, enabling offline distribution of synchronization changes. This feature includes functionality to ensure that the correct data is applied.
- **Server-initiated synchronization** You can initiate MobiLink synchronization from the consolidated database. This means you can push data updates to remote databases, as well as cause remote databases to upload data to the consolidated database.
- **Choice of network protocols** Synchronization can occur over TCP/IP, HTTP, or HTTPS. Palm devices can synchronize through HotSync. Windows Mobile devices can synchronize using ActiveSync.

- **Session-based** All changes can be uploaded in a single transaction and downloaded in a single transaction. At the end of each successful synchronization, the consolidated and remote databases are consistent. (If you want to preserve the order of transactions, you can also choose to have each transaction on the remote database uploaded as a separate transaction.)

  Either a whole transaction is synchronized, or none of it is synchronized. This ensures transactional integrity for each database.

- **Data consistency** MobiLink operates using a loose consistency policy. All changes are synchronized with each site over time in a consistent manner, but different sites may have different copies of data at any instant.

- **Wide variety of hardware and software platforms** A variety of widely-used database management systems can be used as a MobiLink consolidated database, or you can define synchronization to an arbitrary data source using the MobiLink server API. Remote databases can be SQL Anywhere or UltraLite. The MobiLink server runs on Windows, Unix, Linux, and Mac OS X. SQL Anywhere runs on Windows, Windows Mobile, or Unix, Linux, and Mac OS X. UltraLite runs on Palm or Windows Mobile. See "Supported platforms" [*SQL Anywhere 11 - Introduction*].

Quick start to MobiLink

MobiLink is designed to synchronize data among many remote applications that connect intermittently with one or more central data sources. In a basic MobiLink application, your remote clients are SQL Anywhere or UltraLite databases, and your central data source is one of the supported ODBC-compliant relational databases. This architecture can be extended using the MobiLink server API so that there are virtually no restrictions on what you synchronize to on the server side.

In all MobiLink applications, the MobiLink server is the key to the synchronization process. Synchronization typically begins when a MobiLink remote site opens a connection to a MobiLink server. During synchronization, the MobiLink client at the remote site can upload database changes that were made to the remote database since the previous synchronization. On receiving this data, the MobiLink server updates the consolidated database, and then can download changes from the consolidated database to the remote database.

The quickest way to start developing a MobiLink application is to use the **Create Synchronization Model Wizard**. When you use the wizard, most of the steps outlined below are handled for you. See "Introduction to MobiLink models" on page 26.

However, even when using a MobiLink model, you need to understand the process and components of MobiLink synchronization.

Overview of a MobiLink application

To create a MobiLink application

1. Set up a consolidated database.
   - Run setup scripts against the database to add system objects required by MobiLink synchronization. Alternatively, you can create a separate system database to hold these objects.

     See "MobiLink consolidated databases" [*MobiLink - Server Administration*].
2. Set up remote databases.
   - Your remote databases can be SQL Anywhere, UltraLite, or a combination of the two.
   - In your remote databases, create MobiLink users. See "MobiLink users" [*MobiLink - Client Administration*].
   - To determine the upload in a SQL Anywhere remote, create publications and subscriptions. See "Publishing data" [*MobiLink - Client Administration*].

     To determine the upload in an UltraLite remote, create publications. See "Publications in UltraLite" [*UltraLite - Database Management and Reference*].
3. To determine how the upload is applied, create server synchronization logic. See "Introduction to synchronization scripts" [*MobiLink - Server Administration*].
4. To download data that has changed since the last download, set up timestamp based synchronization. See "Timestamp-based downloads" [*MobiLink - Server Administration*].
5. Start the MobiLink server. See "MobiLink server" [*MobiLink - Server Administration*].
6. Initiate synchronization on the client.
   - For SQL Anywhere remotes, see "Initiating synchronization" [*MobiLink - Client Administration*].

- For UltraLite remotes, see "Designing synchronization in UltraLite" [*UltraLite - Database Management and Reference*].

Introductory reading

- "Introducing MobiLink synchronization" on page 3
- "Synchronization techniques" [*MobiLink - Server Administration*]

Tutorials

- "Tutorial: Introduction to MobiLink" on page 51
- "Exploring the CustDB sample for MobiLink" on page 57
- "Exploring the CustDB samples for UltraLite" [*UltraLite - Database Management and Reference*]
- "Exploring the MobiLink Contact sample" on page 77
- "Tutorial: Using MobiLink with an Oracle 10g consolidated database" on page 91
- "Tutorial: Using MobiLink with an Adaptive Server Enterprise consolidated database" on page 111
- "Tutorial: Using Java synchronization logic" on page 131
- "Tutorial: Using .NET synchronization logic" on page 143
- "Tutorial: Using .NET and Java for custom authentication" on page 155
- "Tutorial: Introduction to direct row handling" on page 165
- "Tutorial: Synchronizing with Microsoft Excel" on page 191
- "Tutorial: Synchronizing with XML" on page 213

Other resources for getting started

- MobiLink provides samples that you can examine and run to explore MobiLink functionality. MobiLink samples are installed with the product in *samples-dir\MobiLink*. (For the location of *samples-dir* on your operating system, see "Samples directory" [*SQL Anywhere Server - Database Administration*].)
- MobiLink code exchange samples are located at http://ianywhere.codexchange.sybase.com/servlets/ProjectDocumentList?folderID=319.
- You can post questions on the MobiLink newsgroup: sybase.public.sqlanywhere.mobilink

Designing a MobiLink application

There are two basic architectures for database applications:

- **Online applications** Users update data by connecting to the central database directly. When a connection is unavailable, the user cannot work.
- **Occasionally connected smart client applications** Each user has a local database. Their database application is always available to them, regardless of connectivity, and is kept in sync with other databases in the system.

MobiLink is designed for creating occasionally connected smart client applications. Smart client applications can greatly increase the usability, efficiency, and scalability of an application, but they pose new issues for application developers. This section describes some of the major issues facing developers of smart client applications, and describes how you can implement solutions in a MobiLink synchronization environment.

Synchronize only what you need

In most applications it would be a disaster to download the entire consolidated database every time you want to update any piece of data on your remote device. The time and bandwidth would be prohibitive, making the whole system unworkable. There are various techniques for ensuring you upload and download only what each user needs.

First, each remote database should only contain a subset of the tables and columns in the consolidated database. For example, a salesperson in Region A may need different tables and columns from a salesperson in Region B or a supervisor.

Of the tables and columns that you put on a remote device, you only want to mark ones for synchronization that need to be synchronized. In a MobiLink application you can map tables and columns, regardless of their names, as long as the data types match. By default, data is both uploaded and downloaded, but MobiLink also allows you to specify that certain columns are upload-only or download-only.

Your synchronization should only download rows to a remote database that are relevant to the user. You might want to partition your download by remote database, by user, or by other criteria. For example, a sales rep in Region A may only need data updates about Region A.

You only want to update data that has changed. In a MobiLink application the upload is based on the transaction log and so by default data is only uploaded if it has changed on the remote database. To do the same for the download, you specify timestamp-based synchronization so that your system records the time that data is successfully downloaded, and data is downloaded only if it has changed since then.

You may also want to implement a system of high priority synchronization: time-sensitive data is scheduled to be updated frequently, but less time-critical data is scheduled to be updated at night or when the device is in a cradle. You can implement high-priority synchronization by creating different publications that are scheduled to run at different times.

In addition, your users may benefit from a push-synchronization system, where data is effectively pushed down to remote devices when needed. For example, if a trucking company dispatcher learns of a traffic disruption, they can download an update to the truck drivers who are heading towards that area. In MobiLink, this is called server-initiated synchronization.

Handle upload conflicts

Say you have a warehouse. Each employee has a handheld device that they use to update inventory as they add or remove boxes. They start a shift with 100 boxes, so each employee's remote database registers 100, as does the consolidated database. David removes 20 boxes. He updates his database and synchronizes. Now both his database and the consolidated database register 80. Now Susan removes ten boxes. But when Susan updates her database and synchronizes, her application expects the consolidated database to have 100 boxes, not 80. This generates an upload conflict.

In this warehouse application, the solution is to create conflict resolution logic that says that the correct value is whatever David updated it to, minus the original value less Susan's value:

```
80 - (100 - 90) = 70
```

While this conflict resolution logic works for inventory-based applications such as a warehouse, it isn't appropriate in all business applications. With MobiLink, you can define conflict resolution logic to cover:

- **Inventory model** Update the row for the correct number of units.
- **Date** The latest update wins (based on when the value was changed in the database, not when the value was synchronized).
- **Person** For example, the manager always wins or the owner of the record always wins.
- **Custom** Just about any other business logic you need to implement.

In some cases you can design your system so that upload conflicts can't occur. If data is partitioned on the remotes so that there is no overlap, conflicts may be avoided. However, if conflicts can happen, you should create a programmatic solution for detecting and resolving them.

Unique primary keys

In order to upload data, detect upload conflicts, and synchronize deleted rows on the consolidated database, you must have unique primary keys on every synchronized table in your database system. Each row must have a primary key that is unique not only within the database, but within the entire database system. Primary keys must not be updated.

MobiLink provides several ways to guarantee unique primary keys. One is to set the data type of the primary key to a GUID. GUID, which stands for Globally Unique Identifier, is a 16-byte hexadecimal number. MobiLink provides a NEWID function that causes a GUID to be created automatically for a new row.

Another solution is a composite key. In MobiLink, each remote database has a unique value called a remote ID. Your primary keys could be formed from the remote ID plus a regular primary key, such as an ordinal value.

SQL Anywhere also offers a global autoincrement solution. You declare a column as global autoincrement and then when a row is added, the primary key is automatically created by incrementing the last value. This solution works best when your consolidated database is SQL Anywhere.

Finally, you can create a pool of primary key values that are distributed to remote databases.

How you choose which primary key system to use, like many decisions in developing a synchronization solution, has to do with the level of control you have over the consolidated and remote databases. In many cases, the remote databases must be able to operate without any administration. You may also find that it is

difficult to change the schema on the consolidated database. In addition, your choice of RDBMS for the consolidated database may limit your options, as not all RDBMSs support all features.

Handling deletes

Another issue in a synchronization system is how to handle rows that are deleted from the consolidated database. Say I delete a row from the consolidated database. The next time David synchronizes his remote database, the delete is downloaded—deleting the row from David's database. But what do I do with it on the consolidated database? I can't delete it because I need to download the delete to Susan as well.

Here are two ways you can handle download deletes. First, you can add a status column to each table that indicates whether the row is deleted or not. In this case, the row is never deleted—it is just marked for deletion. (From time to time you can clean up the rows marked for deletion, once you are sure that all the remote databases are up to date.) Alternatively, you can create a shadow table for each table. The shadow table stores the primary key values of deleted rows. When a row is deleted, a trigger populates the shadow table, and the values in the shadow table determine what to delete on the remote database.

Transactions

In a synchronized database system, only database transactions that are committed should be synchronized. In addition, all committed transactions involving data that is to be synchronized should be synchronized, or an error should be generated. This is the default behavior in MobiLink.

You also need to consider the isolation level of the connection to the consolidated database. You need to use an isolation level that provides the best performance possible while ensuring data consistency. Isolation level 0 (READ UNCOMMITTED) is generally unsuitable for synchronization as it can lead to inconsistent data.

By default, MobiLink uses the isolation level SQL_TXN_READ_COMMITTED for uploads, and if possible it uses snapshot isolation for downloads (otherwise it uses SQL_TXN_READ_COMMITTED). Snapshot isolation eliminates the problem of downloads being blocked until transactions are closed on the consolidated database, but not all RDBMSs support it.

Daylight savings time

The annual change to daylight savings time can pose a problem for synchronized databases during the hour that the time changes. In the autumn the time moves back an hour; 2:00 AM becomes 1:00 AM. If you attempt to synchronize between 1:00 AM and 2:00 AM, the timestamp of the synchronization is ambiguous: is it the first 1:15 AM or the second 1:15 AM?

To resolve this problem you can shut down for an hour when the time changes in the autumn, or you can put your consolidated database server on coordinated universal time (UTC) time.

Further reading

- “Synchronization techniques” [*MobiLink - Server Administration*]

Options for developing a MobiLink application

MobiLink provides a variety of ways to develop an application. You can use these methods in alone or in combination.

- **Create Synchronization Model Wizard** The wizard walks you through the development of your application. You start with a central database that has schema, and you can create remote databases and the scripts needed for synchronization. The wizard can also create shadow tables on your consolidated database to handle things like download deletes. When the wizard completes, your synchronization model appears in Model mode, where you can further customize it. There is a **Deploy Synchronization Model Wizard** that creates databases and tables, updates the MobiLink system tables, and creates scripts that run MobiLink utilities.

  Once you have deployed a MobiLink model, if you have further customizations to it you have a choice of making the changes in Model mode or making the changes outside the model, using one of the methods described below. If you make changes outside the model and you redeploy the model, you lose the changes.

  See "MobiLink models" on page 25.

- **Sybase Central Admin mode** The MobiLink plug-in for Sybase Central includes a section called Admin mode where you can update all the elements of your MobiLink application.

  See "Model mode" on page 32.

- **System procedures** When you set up a central database to operate as a consolidated database, system objects are created that are used by MobiLink synchronization. These include MobiLink system tables, where the server side of your MobiLink application is largely stored. They also include system procedures and utilities that you can use to insert MobiLink scripts into your MobiLink system tables, register remote users, and so on. See:

  - "MobiLink server system procedures" [*MobiLink - Server Administration*]
  - "MobiLink utilities" [*MobiLink - Server Administration*]

- **Direct manipulation of MobiLink system tables** Advanced users may want to add, delete, and update data in the MobiLink system tables directly. Doing so requires an advanced understanding of how MobiLink works.

  See "MobiLink server system tables" [*MobiLink - Server Administration*].

Options for writing server-side synchronization logic

MobiLink synchronization scripts can be written in SQL, or they can be written in Java (using the MobiLink server API for Java) or in .NET (using the MobiLink server API for .NET).

SQL synchronization logic is usually best when synchronizing to a supported consolidated database.

Java and .NET are useful if you are synchronizing against something other than a supported consolidated database. They may also be useful if your design is restricted by the limitations of the SQL language or by the capabilities of your database management system, or if you simply want portability across different RDBMS types.

Java and .NET synchronization logic can function just as SQL logic functions. The MobiLink server can make calls to Java or .NET methods on the occurrence of MobiLink events just as it can access SQL scripts on the occurrence of MobiLink events. When you are working in Java or .NET, you can use the events to do some extra processing, but when you are processing scripts for events that directly handle upload or download rows, your implementation must return a SQL string. With the exception of the two events used in direct row handling, uploads and downloads are not directly accessible from Java or .NET synchronization logic: MobiLink executes the string returned by Java or .NET as SQL.

Direct row handling, which uses the events handle_UploadData and handle_DownloadData to synchronize against a data source, **does** directly manipulate the upload and download rows.

Following are some scenarios where you might want to consider writing scripts in Java or .NET:

- **Direct row handling** With Java and .NET synchronization logic, you can use MobiLink to access data from data sources other than your consolidated database, such as application servers, web servers, and files.
- **Authentication** A user authentication procedure can be written in Java or .NET so that MobiLink authentication integrates with your corporate security policies.
- **Stored procedures** If your RDBMS lacks the ability to use user-defined stored procedures, you can create a method in Java or .NET.
- **External calls** If your program calls for contacting an external server midway through a synchronization event, you can use Java or .NET synchronization logic to perform actions triggered by synchronization events. Java and .NET synchronization logic can be shared across multiple connections.
- **Variables** If your database lacks the ability to handle variables, you can create a variable in Java or .NET that persists throughout your connection or synchronization. (Alternatively, with SQL scripts you can use user-defined named parameters, which work with all consolidated database types. See "User-defined named parameters" [*MobiLink - Server Administration*].)

MobiLink server APIs

Java and .NET synchronization logic are available via the MobiLink server APIs. The MobiLink server APIs are sets of classes and interfaces for MobiLink synchronization.

The MobiLink server API for Java offers you:

- Access to the existing ODBC connection to the consolidated database as a JDBC connection.
- Access to alternate data sources using interfaces such as JDBC, web services, and JNI.

- The ability to create new JDBC connections to the consolidated database to make database changes outside the current synchronization connection. For example, you can use this for error logging or auditing, even if the synchronization connection does a rollback.
- For synchronizing with the consolidated database, the ability to write and debug Java code before it is executed by the MobiLink server. SQL development environments for many database management systems are relatively primitive compared to those available for Java applications.
- Both SQL row handling and direct row handling.
- The full richness of the Java language and its large body of existing code and libraries.

See "MobiLink server API for Java Reference" [*MobiLink - Server Administration*].

The MobiLink server API for .NET offers you:

- Access to the existing ODBC connection to the consolidated database using iAnywhere classes that call ODBC from .NET.
- Access to alternate data sources using interfaces such as ADO.NET, web services, and OLE DB.
- For synchronizing with the consolidated database, the ability to write and debug .NET code before it is executed by the MobiLink server. SQL development environments for many database management systems are relatively primitive compared to those available for .NET applications.
- Both SQL row handling and direct row handling.
- Code that runs inside the .NET Common Language Runtime (CLR) and allows access to all .NET libraries, including both SQL row handling and direct row handling.

See "MobiLink server API for .NET reference" [*MobiLink - Server Administration*].

See also

- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "Synchronization techniques" [*MobiLink - Server Administration*]
- "Writing synchronization scripts in Java" [*MobiLink - Server Administration*]
- "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*]
- "Direct row handling" [*MobiLink - Server Administration*]

The synchronization process

A **synchronization** is a process of data exchange between MobiLink clients and a central data source. During this process, the client must establish and maintain a session with the MobiLink server. If successful, the session leaves the remote and consolidated databases in a mutually consistent state.

The client normally initiates the synchronization process. It begins by establishing a connection to the MobiLink server.

The upload and the download

To upload rows, MobiLink clients prepare and send an **upload** that contains a list of all the rows that have been updated, inserted, or deleted on the remote database since the last synchronization. Similarly, to download rows, the MobiLink server prepares and sends a **download** that contains a list of inserts, updates, and deletes.

1. **Upload** By default, the MobiLink client automatically keeps track of which rows in the remote database have been inserted, updated, or deleted since the last successful synchronization. Once the connection is established, the MobiLink client uploads a list of all these changes to the synchronization server.

   The upload consists of a set of new and old row values for rows modified in the remote database. (Updates have new and old row values; deletes have old values; and inserts have new values.) If a row has been updated or deleted, the old values are those that were present immediately following the last successful synchronization. If a row has been inserted or updated, the new values are the current row values. No intermediate values are sent, even if the row was modified several times before arriving at its current state.

   The MobiLink server receives the upload and executes upload scripts that you define. By default it applies all the changes in a single transaction. When it has finished, the MobiLink server commits the transaction.

2. **Download** The MobiLink server compiles a list of rows to insert, update, or delete on the MobiLink client, using synchronization logic that you create. It downloads these rows to the MobiLink client. To compile this list, the MobiLink server opens a new transaction on the consolidated database.

   The MobiLink client receives the download. It takes the arrival of the download as confirmation that the consolidated database has successfully applied all uploaded changes. It ensures that these changes are not sent to the consolidated database again.

   Next, the MobiLink client automatically processes the download, deleting old rows, inserting new rows, and updating rows that have changed. It applies all these changes in a single transaction in the remote database. When finished, it commits the transaction.

During MobiLink synchronization, there are few distinct exchanges of information. The client builds and uploads the entire upload. In response, the synchronization server builds and downloads the entire download. Limiting the chattiness of the protocol is especially important when communication is slower and has higher latency, as is the case when using telephone lines or public wireless networks.

Note
MobiLink operates using the ODBC isolation level SQL_TXN_READ_COMMITTED as the default isolation level for the consolidated database. If the RDBMS used for the consolidated database supports snapshot isolation, and if snapshot is enabled for the database, then by default MobiLink uses snapshot isolation for downloads. See "MobiLink isolation levels" [*MobiLink - Server Administration*].

See also

- "Overview of MobiLink events" [*MobiLink - Server Administration*]
- "Events during upload" [*MobiLink - Server Administration*]
- "Events during download" [*MobiLink - Server Administration*]

MobiLink events

When the MobiLink client initiates a synchronization, a number of synchronization events occur. At the occurrence of a synchronization event, MobiLink looks for a script to match the event. The script contains instructions detailing what you want done. If you have defined a script for the event and put it in a MobiLink system table, it is invoked.

MobiLink scripts

Whenever an event occurs, the MobiLink server executes the associated script if you have created one. If no script exists, the next event in the sequence occurs.

Note
When you use the **Create Synchronization Model Wizard** to create your MobiLink application, all the required MobiLink scripts are created for you. However, you can customize the default scripts, including creating new scripts.

Following are the typical upload scripts for tables. The first event, upload_insert, triggers the running of the upload_insert script, which inserts any changes in the emp_id and emp_name columns into the emp table. The upload_delete and upload_update scripts perform similar functions for delete and update actions on the emp table.

Event	Example script contents

upload_insert	`INSERT INTO`
	`emp (emp_id,emp_name)`
	`VALUES {ml r.emp_id}, {ml r.emp_name}`

upload_delete	`DELETE FROM emp` `WHERE emp_id = {ml r.emp_id}`

upload_update	`UPDATE emp`
	`SET emp_name = {ml r.emp_name}`
	`WHERE emp_id = {ml r.emp_id}`

The download script uses a cursor. Following is an example of a download_cursor script:

```
SELECT order_id, cust_id
FROM ULOrder
WHERE last_modified >= {ml s.last_table_download}
AND emp_name = {ml r.emp_id}
```

For more information about events and scripts, see:

- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "Synchronization events" [*MobiLink - Server Administration*]

Scripts can be written in SQL, Java, or .NET

You can write scripts using the native SQL dialect of your consolidated database, or using Java or .NET synchronization logic. Java and .NET synchronization logic allow you to write code, invoked by the MobiLink server, to connect to a database, manipulate variables, directly manipulate uploaded row operations, or add row operations to the download. There is a MobiLink server API for Java and a MobiLink server API for .NET that provide classes and methods to suit the needs of synchronization.

See "Options for writing server-side synchronization logic" on page 14.

For information about RDBMS-dependent scripting, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

Storing scripts

SQL scripts are stored in MobiLink system tables in the consolidated database. For scripts written with the MobiLink server APIs, you store the fully qualified method name as the script. You can add scripts to a consolidated database in several ways:

- When you use the **Create Synchronization Model Wizard**, scripts are stored in the MobiLink system tables when you deploy your project.
- You can manually add scripts to the system tables by using stored procedures that are installed when you set up a consolidated database.
- You can manually add scripts to the system tables using Sybase Central.

See "Adding and deleting scripts" [*MobiLink - Server Administration*].

Transactions in the synchronization process

The MobiLink server incorporates changes uploaded from each MobiLink client into the consolidated database in one transaction. It commits these changes once it has completed inserting new rows, deleting old rows, making updates, and resolving any conflicts.

The MobiLink server prepares the blocking download, including all deletes, inserts, and updates, using another transaction. If you specify download acknowledgement and the client confirms a successful download, the MobiLink server commits the download transaction. If the application encounters problems or cannot reply when blocking download acknowledgement is specified, the MobiLink server instead rolls back the download transaction. The default is to not use download acknowledgement.

There should be no implicit or explicit commit or rollback in your SQL synchronization scripts or the procedures or triggers that are called from your SQL synchronization scripts. COMMIT or ROLLBACK statements within SQL scripts alter the transactional nature of the synchronization steps. If you use them, MobiLink cannot guarantee the integrity of your data in the event of a failure.

Tracking downloaded information

MobiLink uses a last download timestamp, stored in the remote database, to help simplify how downloads are created.

The primary role of the download transaction is to select rows in the consolidated database. If the download fails, the remote uploads the same last download timestamp over again, and no data is lost.

See "Using last download times in scripts" [*MobiLink - Server Administration*].

Begin and end transactions

The MobiLink client processes information in the download in one transaction. Rows are inserted, updated, and deleted to bring the remote database up to date with the consolidated data.

The MobiLink server uses two other transactions, one at the beginning of synchronization and one at the end. These transactions allow you to record information regarding each synchronization and its duration. Thus, you can record statistics about attempted synchronizations, successful synchronizations, and the duration of synchronizations. Since data is committed at various points in the process, these transactions also let you commit data that can be useful when analyzing failed synchronization attempts.

See also

- "Overview of MobiLink events" [*MobiLink - Server Administration*]
- "Events during upload" [*MobiLink - Server Administration*]
- "Events during download" [*MobiLink - Server Administration*]

How synchronization failure is handled

MobiLink synchronization is fault tolerant. For example, if a communication link fails during synchronization, both the remote database and the consolidated database are left in a consistent state.

On the client, failure is indicated by a return code.

Synchronization failure is handled differently depending on when it happens. The following cases are handled in different ways:

- **Failure during upload** If the failure occurs while building or applying the upload, the remote database is left in exactly the same state as at the start of synchronization. At the server, any part of the upload that has been applied is rolled back.
- **Failure between upload and download** If the failure occurs once the upload is complete, but before the MobiLink client receives the download, the client cannot be certain whether the uploaded changes were successfully applied to the consolidated database. The upload might be fully applied and committed,

or the failure may have occurred before the server applied the entire upload. The MobiLink server automatically rolls back incomplete transactions in the consolidated database.

The MobiLink client maintains a record of all uploaded changes in case they must be sent again. The next time the client synchronizes, it requests the state of the previous upload before building the new upload. If the previous upload was not committed, the new upload contains all changes from the previous upload.

- **Failure during download** When a failure occurs on the remote device while applying the download, any part of the download that has been applied is rolled back and the remote database is left in the same state as before the download.

  If you are using blocking download acknowledgement, the MobiLink server also rolls back the download transaction in the consolidated database.

  If you are using non-blocking download acknowledgement, the download transaction has already been committed, but neither the nonblocking_download_ack script nor the publication_nonblocking_download_ack script is invoked.

  If you are not using download acknowledgement, there is no server-side consequence of a download failure.

In all cases where failure may occur, no data is lost. The MobiLink server and the MobiLink client manage this for you. The developer or user do not need to worry about maintaining consistent data in their application.

How the upload is processed

When the MobiLink server receives an upload from a MobiLink client, the entire upload is stored until the synchronization is complete. This is done for the following reasons:

- **Filtering download rows** The most common technique for determining which rows to download is to download rows that have been modified since the most recent download. When synchronizing, the upload precedes the download. Any rows that are inserted or updated during the upload are rows that have been modified since the previous download.

  It would be difficult to write a download_cursor script that omits from the download rows that were sent as part of the upload. For this reason, the MobiLink server automatically removes these rows from the download.

- **Processing inserts and updates** By default, tables in the upload are applied to the consolidated database in an order that avoids referential integrity violations. The tables in the upload are ordered based on foreign key relationships. For example, if table A and table C both have foreign keys that reference a primary key column in B, then inserts and updates for table B rows are uploaded first.

- **Processing deletes after inserts and updates** Deletes are applied to the consolidated database after all inserts and updates are applied. When deletes are applied, tables are processed in the opposite order from the way they appear in the upload. When a row being deleted references a row in another table that is also being deleted, this order of operations ensures that the referencing row is deleted before the referenced row is deleted.

- **Deadlock** When an upload is being applied to the consolidated database, it may encounter deadlock due to concurrency with other transactions. These transactions might be upload transactions from other MobiLink server database connections, or transactions from other applications using the consolidated database. When an upload transaction is deadlocked, it is rolled back and the MobiLink server automatically starts applying the upload again, from the beginning.

Performance tip
It is important to write your synchronization scripts to avoid contention as much as possible. Contention has a significant impact on performance when multiple users are synchronizing simultaneously.

Referential integrity and synchronization

All MobiLink clients enforce referential integrity when they incorporate the download into the remote database.

Rather than failing the download transaction, by default the MobiLink client automatically deletes all rows that violate referential integrity.

This feature has the following benefits:

- Protection from mistakes in your synchronization scripts. Given the flexibility of the scripts, it is possible to accidentally download rows that would break the integrity of the remote database. The MobiLink client automatically maintains referential integrity without requiring intervention.
- You can use this referential integrity mechanism to delete information from a remote database efficiently. By only sending a delete to a parent record, the MobiLink client removes all the child records automatically for you. This can greatly reduce the amount of traffic MobiLink must send to the remote database.

MobiLink clients provide notification if they have to explicitly delete rows to maintain referential integrity, as follows:

- For SQL Anywhere clients, dbmlsync writes an entry in the log. There are also dbmlsync event hooks that you can use. See:
  - "sp_hook_dbmlsync_download_ri_violation" [*MobiLink - Client Administration*]
  - "sp_hook_dbmlsync_download_log_ri_violation" [*MobiLink - Client Administration*]
- For UltraLite clients, the warning SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY is raised. This warning takes a parameter which is the table name. To maintain referential integrity, the warning is raised on every row that is deleted. Your application can ignore the warnings if you want synchronization to proceed. If you want to explicitly handle the warnings, you can use the error callback function to trap them and, for example, count the number of rows that are deleted.

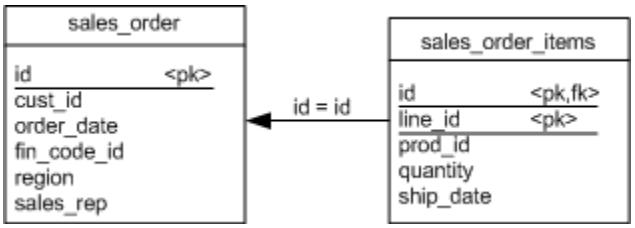
  If you want synchronization to fail when the warning is raised, you must implement a synchronization observer and then signal the observer (perhaps through a global variable) from the error callback function. In this case, synchronization fails on the next call to the observer.

Referential integrity checked at the end of the transaction

The MobiLink client incorporates changes from the download in a single transaction. To offer more flexibility, referential integrity checking occurs at the end of this transaction. Because checking is delayed, the database may temporarily pass through states where referential integrity is violated. This is because rows that violate referential integrity are automatically removed before the download is committed.

Example

Suppose that an UltraLite sales application contains the following two tables. One table contains sales orders. Another table contains items that were sold in each order. They have the following relationship:



If you use the download_delete_cursor for the sales_order table to delete an order, the default referential integrity mechanism automatically deletes all rows in the sales_order_items table that point to the deleted sales order.

This arrangement has the following advantages:

- You do not require a sales_order_items table script because rows from this table are deleted automatically.
- The efficiency of synchronization is improved. You need not download rows to delete from the sales_order_items table. If each sales order contains many items, the performance improves because the download is now smaller. This technique is particularly valuable when using slow communication methods.

Changing the default behavior

For SQL Anywhere clients, you can use the sp_hook_dbmlsync_download_ri_violation client event hook to handle the referential integrity violation. Dbmlsync also writes an entry to its log.

See:

- "sp_hook_dbmlsync_download_log_ri_violation" [*MobiLink - Client Administration*]
- "sp_hook_dbmlsync_download_ri_violation" [*MobiLink - Client Administration*]

Security

There are several aspects to securing data throughout a widely distributed system such as a MobiLink installation:

- **Protecting data in the consolidated database** Data in the consolidated database can be protected using the database user authentication system and other security features.

  For more information, see your database documentation. If you are using a SQL Anywhere consolidated database, see "Keeping your data secure" [*SQL Anywhere Server - Database Administration*].

- **Protecting data in the remote databases** If you are using SQL Anywhere remote databases, the data can be protected using SQL Anywhere security features. By default, these features are designed to prevent unauthorized access through client/server communications, but not to be proof against a serious attempt to extract information directly from the database file.

  Files on the client are protected by the security features of the client operating system.

  If you are using a SQL Anywhere remote database, see "Keeping your data secure" [*SQL Anywhere Server - Database Administration*].

  If you are using an UltraLite database, see "Securing UltraLite databases" [*UltraLite - Database Management and Reference*].

- **Protecting data during synchronization** Communication from MobiLink clients to MobiLink servers can be protected by the MobiLink transport layer security features. See "Transport-layer security" [*SQL Anywhere Server - Database Administration*].

- **Protecting the synchronization system from unauthorized users** MobiLink synchronization can be secured by a password-based user authentication system. This mechanism prevents unauthorized users from synchronizing data. See "MobiLink users" [*MobiLink - Client Administration*].

CHAPTER 2

MobiLink models

Contents

Introduction to MobiLink models

A **synchronization model** is a tool that makes it easy for you to create MobiLink applications. A synchronization model is a file that is created by the **Create Synchronization Model Wizard** in Sybase Central.

When you run the **Create Synchronization Model Wizard**, you are prompted to connect to a consolidated database to obtain schema information. If your database is not yet set up as a consolidated database, the wizard can apply setup scripts to create MobiLink system tables and other objects required by synchronization; other than that, no changes are made to your consolidated database until you deploy your model. When you complete the wizard the connection to the database is closed.

After you complete the **Create Synchronization Model Wizard**, the model appears in **Model mode**. You can use **Model** mode to customize your model. When you are in **Model** mode, you are working offline: no changes are made to your consolidated database. Your model is stored in a model file with the extension *.mlsm*.

When your model is complete, you use the **Deploy Synchronization Model Wizard** to deploy it. The **Deploy Synchronization Model Wizard** creates script files to run the synchronization server and client using deployment options you choose. You can choose to make changes to your existing databases when you deploy or you can choose to have the wizard create files that you run yourself.

After you deploy, you can continue to customize the model or databases and then redeploy. Alternatively, you can modify the deployed synchronization system out of **Model** mode, using the techniques that are described throughout the MobiLink documentation. However, when you change a synchronization system outside of **Model** mode, you cannot reverse-engineer the changes back into the model.

Limitations of MobiLink models

Following are some restrictions when you use the **Create Synchronization Model Wizard** and **Model** mode:

- **Changes made outside the model cannot be redeployed** If you deploy a model and then make changes to it outside the model, those changes are not saved in the model. This is fine if you want to use the model as a starting point, deploy, and then make all your changes outside the model. But if you want to be able to redeploy the model, you are better off making your changes in **Model** mode so that they are saved and can be redeployed.
- **DB2 mainframe** MobiLink models do not work with DB2 mainframe consolidated databases.
- **MobiLink system database** You cannot use a MobiLink system database.
- **Multiple publications** You cannot create multiple publications. After you have deployed your model you can add more publications using non-model methods such as the CREATE PUBLICATION statement, but you cannot reverse-engineer these additions back into your model.
- **Views** When you are selecting consolidated database tables for table mappings, it is not possible to select a view.

- **Generated remote database** If you create a new remote schema in the **Create Synchronization Model Wizard**, the new remote database columns do not contain the foreign keys, indexes, or default column values of the columns in the consolidated database. UltraLite databases do not support NCHAR or NVARCHAR columns, so you cannot use consolidated database tables with columns of those data types to generate a new remote schema for an UltraLite remote database. After deploying, you can create or change the remote schema and then update the schema for the model.
- **Computed columns** If you want to synchronize a consolidated database table that has computed columns, you cannot upload to the table. If you deploy a synchronization model with computed columns, the deployment may have errors creating the trigger used for timestamp-based downloads. You can either exclude the column from synchronization, or configure the table as download-only (and either use snapshot download or edit the generated consolidated SQL file to remove the computed column from the trigger definition).

  The Microsoft SQL Server AdventureWorks sample database contains computed columns. If you use this database to create a model, set the columns to be download-only or exclude the columns from synchronization.
- **Logical deletes** The MobiLink synchronization model support for logical deletes assumes that a logical delete column is only on the consolidated database and not on the remote. When copying a consolidated schema to a new remote schema, leave out any columns that match the logical delete column in the model's synchronization settings. For a new model, the default column name is deleted.

  To add the logical delete column name to the remote schema:

  1. In the wizard, choose **Use logical deletes**.
  2. Rename the logical delete column so that it does not match any column names in the consolidated.
  3. When the wizard is finished, update the remote schema and keep the default table selection. The logical delete column name appears in the schema change list and be added to remote schema.

> **Note**
> You need to set the column mapping for the remote's logical delete column to the consolidated's logical delete column.

Deployment considerations

- **Long object names** The database objects that are created when deploying may have names that are longer than the database supports (because the new object names are created by adding suffixes to the base table names). If this happens, deploy only to file (not directly to a database) and edit the generated SQL file to replace all occurrences of the name that is too long.
- **New remote schemas** If you create a new remote schema in the **Create Synchronization Model Wizard**, the new remote database columns do not contain the foreign keys, indexes, or default column values of the columns in the consolidated database.

  UltraLite databases do not support NCHAR or NVARCHAR columns, so you cannot use consolidated database tables with columns of those data types to generate a new remote schema for an UltraLite remote database.

  After deploying, you can create or change the remote schema and then update the schema for the model.

- **Proxy tables** It is possible to synchronize with consolidated database tables that proxy tables to another database, but if you use a TIMESTAMP column for timestamp-based downloads, then you need to add the TIMESTAMP column to both the base table and the proxy table. The **Deploy Synchronization Model Wizard** cannot add a column to a proxy table or its base, so you either need to use an existing column on both the base and proxy, or you need to use a shadow table or snapshot download.
- **Materialized views** If you are using timestamp-based downloads and have chosen to add a timestamp column to consolidated tables, you must disable any materialized views that depend on the tables before deploying. Otherwise you may get errors when trying to alter the tables. For SQL Anywhere consolidated databases, use the sa_dependent_views system procedure to find out if a table has dependent materialized views. See sa_dependent_views system procedure.

Other considerations

- **Creating remote databases based on an Oracle consolidated database** When you are using an Oracle consolidated database as the basis for your SQL Anywhere or UltraLite remote database, you may want to change DATE columns in the consolidated database to TIMESTAMP. Otherwise, sub-second information is lost on upload.
- **Proxy tables** If you are using shadow tables maintained by triggers, then the triggers and shadow tables should be on the base table. A trigger on a base table can't modify a shadow table in the database that defines the proxy table for the base table.

Creating models

You create MobiLink models with the **Create Synchronization Model Wizard**.

Create Synchronization Model Wizard

The **Create Synchronization Model Wizard** takes you through the steps of creating a MobiLink synchronization application.

Overview of setting up a MobiLink application with the Create Synchronization Model Wizard

1. Open Sybase Central:

   Choose **Programs** » **SQL Anywhere 11** » **Sybase Central**.

2. From the **Tools** menu, choose **MobiLink 11** » **Set Up MobiLink Synchronization**.

   The **Create Synchronization Model Wizard** appears.

3. Choose a name and location for your model. Your model is stored in a model file with the extension *.mlsm*.

   Note: The name and location are used for default names for files and directories that are created by the wizard.

4. The **Primary Key Requirements** page appears.

   This page is included for safety. It reminds you to maintain unique, permanent primary keys. To continue, you must select the three checkboxes. You can disable this page for the future by selecting the box at the bottom of the page.

   For more information, see "Maintaining unique primary keys" [*MobiLink - Server Administration*].

5. The **Consolidated Database Schema** page appears. You must connect to the consolidated database in your MobiLink application so that the wizard can obtain schema information for it.

   If this database has not been set up for use as a consolidated database, the wizard prompts you to do so. The MobiLink setup process adds system objects to the database that are required by MobiLink. If you choose, these objects are added to the consolidated database immediately. (You can choose to do this setup later, either in the **Deploy Synchronization Model Wizard** or by applying setup files yourself.)

   For more information, see "Setting up a consolidated database" [*MobiLink - Server Administration*].

   The connection to the consolidated database is closed if you choose a different consolidated database, or when you exit the wizard. From this point on, changes you make in this wizard are made to the model file, not to the consolidated database.

6. The **Remote Database Schema** page appears. You can create your remote database schema based on the consolidated database or an existing remote database. The existing remote database can be SQL Anywhere or UltraLite. (When you deploy, you can apply the schema to a new or existing remote database. See "Remote database" on page 30.)

For a new SQL Anywhere remote database, the owner of the remote tables is the same as the owner of the corresponding consolidated database tables. If you want a different owner, you should instead use an existing remote database with the table ownership you set up.

7. Follow the remaining instructions in the wizard. Default recommendations based on best practices are used where possible. There is online help for all the pages.
8. Click **Finish**.

   When you click **Finish**, the model you just created opens in **Model** mode. At the same time, your connection to the consolidated database is closed. You are now working offline and you can make changes to the model. No changes are made outside the model until you deploy the model: the consolidated database does not change and the remote database is not created or changed until that time.

   See "Model mode" on page 32 and "Deploying models" on page 43.

Notes

- A model can have only one publication. See "Publishing data" [*MobiLink - Client Administration*].
- A model can have only one version. See "Script versions" [*MobiLink - Server Administration*].

Remote database

The model contains schema for the remote database. This schema can be obtained from an existing remote database or from the consolidated database.

Use an existing remote database in the following cases:

- If you already have a remote database, especially if its schema is not a subset of the consolidated database schema.
- If your consolidated and remote columns need to have different types. For example, if you need to map an NCHAR column on the consolidated to a CHAR column on an UltraLite remote.
- If your remote tables need to have different owners from the tables on the consolidated database. For a new SQL Anywhere remote database, the owner of the remote tables is the same as the owner of the corresponding consolidated database tables. If you want a different owner, you should use an existing SQL Anywhere remote database with table ownership you set up. (UltraLite databases do not have owners.)

Tip
If you need to change the schema of an existing remote database, make the changes to the database outside of the model, and then run the **Update Schema Wizard** to update the model.

When you deploy your model, you have three options for your remote database, regardless of how you created the remote schema in the model. Your deploy-time options for the remote database are:

- **Create a new remote database** Deployment can create a new remote database using the remote schema from the synchronization model.

- **Update an existing remote database that has no user tables** If you deploy to an empty remote database, then the remote schema from the model is created in the database. This option is useful when you want to use non-default database creation options, such as collation.

  For SQL Anywhere databases, you can see a list of options that cannot be set after the database is created in the Remarks section of "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

  For UltraLite databases, database properties cannot be changed after the database is created. See "Choosing database creation parameters for UltraLite" [*UltraLite - Database Management and Reference*].

- **Update an existing remote database that has a schema matching the schema in the model** This is useful when you have an existing remote database that you want to synchronize. When you deploy directly to an existing remote database, no changes are made to any existing remote data. If you try to deploy directly to an existing remote database whose schema doesn't match the remote schema in the model, you are prompted to update the remote schema in the model.

For a SQL Anywhere remote database, tables have the same owners as the original database. (UltraLite tables do not have owners.)

See also

- "Creating models" on page 29
- "Deploying models" on page 43

Changing your consolidated database

Before you can use a database as a MobiLink consolidated database, you must add objects such as tables, columns, and triggers, that are required for synchronization. This is done by running a setup script against the database. There is a separate setup script for each supported RDBMS. These scripts are all located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation. You can verify exactly what the script does by opening it in a text editor.

When you create a Mobilink synchronization model in Sybase Central, the software can run the script for you, if you choose, or you can run it yourself. You are prompted to install MobiLink setup when you first start the **Create Synchronization Model Wizard**. If you do not install it, you are prompted again when you deploy the model.

See also

- "Setting up a consolidated database" [*MobiLink - Server Administration*]
- "Deploying models" on page 43

Model mode

When you complete the **Create Synchronization Model Wizard** or when you open an existing model, the model appears in **Model** mode. You can use **Model** mode to further customize your model. When you are working in **Model** mode, you are offline and the changes you make are made to the model file. No changes are made to your consolidated or remote databases until you deploy.

Admin mode

The MobiLink plug-in for Sybase Central includes two modes, **Model** mode and **Admin** mode. There is a **Mode** menu in the toolbar for switching between the two modes.

You can customize your synchronization application in **Admin** mode. However, if you deploy a model and then make changes outside the model, you cannot reverse-engineer the changes back into the model. So if you plan to redeploy a model, do not change it in **Admin** mode.

For more information about **Admin** mode, see the relevant sections of the documentation.

Modifying MobiLink applications outside of Sybase Central

You can also modify your deployed model outside of Sybase Central.

For example, you can add or modify MobiLink scripts using system procedures. See "MobiLink system procedures" [*MobiLink - Server Administration*].

When you make changes outside of Sybase Central, the same rules apply as for **Admin** mode: you cannot reverse-engineer changes back into the model.

Modifying table and column mappings

Table mappings indicate which tables should be synchronized, how tables should be synchronized, and how the synchronized data should be mapped between the remote and consolidated databases.

Upload-only, download-only, and non-synchronized tables or columns

By default, MobiLink does a complete, bi-directional synchronization. You can change each table to be upload-only or download-only. You can also choose to not synchronize a table.

In a model, you can only specify tables as download-only; you cannot create download-only publications. This is because a model can have only one publication.

To change the table mapping direction

1. In **Model** mode, open the **Mappings** tab.
2. In the **Table Mappings** pane, select a remote table.
3. In the **Mapping Direction** dropdown list, select one of the following:
   - **Bi-directional**
   - **Upload to consolidated only**

- **Download to remote only**
- **Not synchronized**

> **Caution**
> By default, shadow tables are not synchronized. Do not attempt to synchronize them.

4. If necessary, select a consolidated table to map to from the **Consolidated Table** dropdown list.

To not synchronize a column

1. In **Model** mode, open the **Mappings** tab.
2. In the **Table Mappings** pane, select a table.

   Column information for the table appears in the **Column Mappings** tab in the lower pane.
3. Select a column.
4. In the **Mapping Direction** dropdown list, choose **Not Synchronized**.

   Primary keys must be synchronized.

Changing table and column mappings

If your model is based on an existing remote database, the column mappings represent a best guess. You should check them and customize them as required.

To change table mappings

1. In **Model** mode, open the **Mappings** tab.
2. In the **Table Mappings** pane, select a table.
3. To change the consolidated table that is mapped: From the **Consolidated Table** dropdown list, select a different table.
4. To change the remote table that is mapped:
   - If your remote database is based on the consolidated database, use the **Create New Remote Table** window. See "Creating remote tables when the remote database schema is based on the consolidated database" on page 34.
   - If your remote database is based on an existing remote database: from the **Remote Table** dropdown list, select a different table.
5. To change column mappings for a table: Select the table, and open the **Column Mappings** tab in the lower pane.

Modifying the remote database that your model creates

You can modify the schema of the remote database in the model, as follows.

Creating remote tables when the remote database schema is based on the consolidated database

To add tables to the remote database schema in the model, use the **Create New Remote Tables** window. The tables are added to the model and are mapped for synchronization. To open the **Create New Remote Tables** window in **Model** mode, choose **File** » **New** » **Remote Tables**.

If you want to add a table to the remote database and the table does not exist on the consolidated database, add the table on the consolidated database, run the **Update Schema Wizard**, and then use the **Create New Remote Tables** window to add the table to the model. To open the **Update Schema Wizard** in **Model** mode, from the **File** menu, choose **Update** Schema.

See "Updating schemas in Model mode" on page 42.

Creating remote tables when the remote database schema is based on an existing remote database

If you want to create new tables for an existing remote database, modify the remote database outside of the model and then use the **Update Schema Wizard** to update the remote database schema in the model. You then need to map the new remote tables in the **Mappings** tab. See:

- "Updating schemas in Model mode" on page 42
- "Modifying table and column mappings" on page 32

Deleting remote tables and columns

Model mode allows you to delete tables and columns from the remote database schema that is in the model. You can mark a remote table or column for deletion by right-clicking the row and choosing **Delete**. The remote table or column is deleted from the schema when you save the model. Deleting a remote table or column means that it is not created in the remote database when you deploy to a new remote database.

You cannot delete a primary key.

You cannot delete tables or rows from the consolidated database in **Model** mode. To change the consolidated schema, modify the consolidated database outside of **Model** mode and then run the **Update Schema Wizard**.

Modifying the download type

The download type can be timestamp, snapshot, or custom. You change the download type in the Table Mappings pane of the **Mappings** tab.

- **Timestamp-based download** Choose this option to use timestamp-based download as the default. Only rows that have been changed since the last synchronization are downloaded. See "Timestamp-based downloads" [*MobiLink - Server Administration*].
- **Snapshot download** Choose this option to use snapshot download as the default. All rows are downloaded even if they have not been changed since the last synchronization. See:
  - "Snapshot synchronization" [*MobiLink - Server Administration*]
  - "When to use snapshot synchronization" [*MobiLink - Server Administration*]
- **Custom download logic** Choose this option if you want to write your own download_cursor and download_delete_cursor scripts instead of having them generated for you. See:

- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "Writing download_cursor scripts" [*MobiLink - Server Administration*]
- "Writing download_delete_cursor scripts" [*MobiLink - Server Administration*]

To change the download type

1. In **Model** mode, open the **Mappings** tab.
2. In the **Table Mappings** pane, select a remote table.
3. In the **Download Type** dropdown list, select **Timestamp**, **Snapshot**, or **Custom**.
4. If you chose **Custom**, open the **Events** tab:
   - Right-click the table and choose **Go To Events**.
5. Edit your download_cursor script and download_delete_cursor script with the appropriate business logic.

Modifying how deletes are handled

If you are using snapshot download, all rows in the remote database are deleted before the snapshot is downloaded. If you are using timestamp-based download, you can decide how you want deletes on the consolidated database to be handled on the remote database.

If you want to delete rows from remote databases when they are deleted from the consolidated database, you need to keep a record of the row in order to delete it. You can do this with shadow tables or by using logical deletes.

To change how deletes are handled

1. In **Model** mode, open the **Mappings** tab.
2. In the **Table Mappings** pane, select a remote table.
3. In the **Del.** dropdown list, select the checkbox if you want to download deletes from the consolidated database. Clear the checkbox if you do not want to download deletes from the consolidated database.
4. If you chose to download deletes, open the **Download Deletes** tab in the lower pane.

   To record deletions, you can choose to use a shadow table or logical deletes.

See also

- "Handling deletes" [*MobiLink - Server Administration*]
- "Writing download_delete_cursor scripts" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]

Modifying the download subset

Each MobiLink remote database can synchronize a subset of the data in the consolidated database. You can customize the download subset for each table.

The download subset options are:

- **User** Choose this option to partition data by MobiLink user name, which downloads different data to different registered MobiLink users.

  To use this option, the MobiLink user names must be on your consolidated database. You choose your MobiLink user names when you deploy, so you can choose names that match existing values on your consolidated database. (The column you use for MobiLink user names must be of a type that can hold the values you are using for the user name.) If the MobiLink user names are in a different table from the one you are subsetting, you must join to that table.

- **Remote** Choose this option to partition data by remote ID, which downloads different data to different remote databases.

  To use this option, the remote IDs must be on your consolidated database. Remote IDs are created as GUIDs by default, but you can set the remote IDs to match existing values on your consolidated database. (The column you use for remote IDs must be of a type that can hold the values you are using for the remote IDs.) If the remote IDs are in a different table from the one you are subsetting, you must join to that table.

- **Custom** Choose this option to use a SQL expression that determines which rows are downloaded. Each synchronization only downloads rows where your SQL expression is true. This SQL expression is the same as is used in download_cursor scripts. It is partly generated for you.

To change the download subset

1. In **Model** mode, open the **Mappings** tab.
2. In the **Table Mappings** pane, select a remote table.
3. In the **Dnld Sub.** dropdown list, choose one of the following download subsets: **None**, **User**, **Remote**, or **Custom**.
4. If you chose **User**, **Remote**, or **Custom**, open the **Download Subset** tab in the lower pane.
5. If you chose **User** or **Remote**, the **Download Subset** tab allows you to identify the column in the consolidated table that contains the MobiLink user names or remote IDs, or to enter a join of tables to obtain the MobiLink user names or remote IDs.
6. If you choose **Custom**, the **Download Subset** tab has two text boxes where you add information to construct a download_cursor script. You do not have to write a complete download_cursor. You only need to add extra information to identify the join and other restrictions on the download subset.
   - In the first text box (**Tables to add to the download cursor's FROM clause**), enter the table name(s) if your download_cursor requires a join to other tables. If the join requires multiple tables, separate them with commas.
   - In the second box (**SQL expression to use in the download cursor's WHERE clause**), enter a WHERE condition that specifies the join and the download subset.

See also

- "Introduction to MobiLink users" [*MobiLink - Client Administration*]
- "Remote IDs" [*MobiLink - Client Administration*]
- "Partitioning rows among remote databases" [*MobiLink - Server Administration*]
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]
- "Using last download times in scripts" [*MobiLink - Server Administration*]

- "Writing download_cursor scripts" [*MobiLink - Server Administration*]

Example (User)

For example, the ULOrder table in CustDB can be shared between users. By default, orders are assigned to the employee who created them. But there are times when another employee needs to see orders created by someone else. For example, a manager may need to see all the orders created by employees in their department. The CustDB database has a provision for this via the ULEmpCust table. It allows you to assign customers to employees. They download all orders for that employee customer relationship.

To see how this is done, first view the download_cursor script for ULOrder without download subsetting. Select the ULEmpCust table in the **Mapping** tab. Choose Timestamp-based for the **Dnld. Type** column and None for the **Dnld. sub** column. Right-click the table and choose **Go To Events**. The download_cursor for the table looks like this:

```
SELECT "DBA"."ULOrder"."order_id",
 "DBA"."ULOrder"."cust_id",
 "DBA"."ULOrder"."prod_id",
 "DBA"."ULOrder"."emp_id",
 "DBA"."ULOrder"."disc",
 "DBA"."ULOrder"."quant",
 "DBA"."ULOrder"."notes",
 "DBA"."ULOrder"."status"
FROM "DBA"."ULOrder"
WHERE "DBA"."ULOrder"."last_modified" >= {ml s.last_table_download}
```

Now go back to the **Mappings** tab. Change the **Dnld. sub** column for ULOrder to User. Open the **Download Subset** tab in the lower pane. Select **Use a column in a joined relationship table**. For the table to join, select ULEmpCust. For the column to match, select emp_id. The join condition should be emp_id = emp_id.

Right-click the table in the top pane and choose **Go To Events**. The download_cursor for the table now looks like this:

```
SELECT "DBA"."ULOrder"."order_id",
   "DBA"."ULOrder"."cust_id",
   "DBA"."ULOrder"."prod_id",
   "DBA"."ULOrder"."emp_id",
   "DBA"."ULOrder"."disc",
   "DBA"."ULOrder"."quant",
   "DBA"."ULOrder"."notes",
   "DBA"."ULOrder"."status"
FROM "DBA"."ULOrder", "DBA"."ULEmpCust"
WHERE "DBA"."ULOrder"."last_modified" >= {ml s.last_table_download}
AND "DBA"."ULOrder"."emp_id" = "DBA"."ULEmpCust"."emp_id"
AND "DBA"."ULEmpCust"."emp_id" = {ml s.username}
```

Example (Custom)

For example, assume you want to subset the download of a table called Customer by MobiLink user and you also want to only download rows where active=1. The MobiLink user names do not exist in the table you are subsetting, so you need to create a join to a table called SalesRep, which contains the user names.

In the **Mappings** tab, Choose **Timestamp-based** for the **Dnld. Type** column and **Custom** for the **Dnld. Sub** column of the Customer table. Open the **Download Subset** tab in the lower pane. In the first box (**Tables to add to the download cursor's FROM clause**), type:

```
SalesRep
```

In the second box (**SQL expression to use in the download cursor's WHERE clause**), type:

```
SalesRep.ml_username = {ml s.username}
   AND Customer.active = 1
   AND Customer.cust_id = SalesRep.cust_id
```

Right-click the table in the top pane and choose **Go To Events**. The download_cursor for the table now looks like this:

```
SELECT "DBA"."Customer"."cust_id",
 "DBA"."Customer"."cust_name"
FROM "DBA"."Customer", SalesRep
WHERE "DBA"."Customer"."last_modified" >= {ml s.last_table_download}
 AND SalesRep.ml_username = {ml s.username}
 AND Customer.active = 1
 AND Customer.cust_id = SalesRep.cust_id
```

The final line of the WHERE clause creates a key join of Customer to SalesRep.

Modifying conflict detection and resolution

When a row is updated on both the remote and consolidated databases, a conflict occurs the next time the databases are synchronized.

You have the following options for detecting conflicts:

- **No conflict detection** Choose this option if you do not want any conflict detection. Uploaded updates are applied without checking for conflicts. This avoids having to fetch current row values from the consolidated database, so the synchronization of updates may be faster.
- **Row-based conflict detection** A conflict is detected if the row has been updated by both the remote and consolidated databases since the last synchronization.

  This option defines an upload_fetch script and upload_update script. See "Detecting conflicts with upload_fetch scripts" [*MobiLink - Server Administration*].
- **Column-based conflict detection** A conflict is detected if the same column has been updated for the row in both the remote and consolidated databases.

  This option defines an upload_fetch_column_conflict script. See "Detecting conflicts with upload_fetch scripts" [*MobiLink - Server Administration*].

  If a table has BLOBs and you choose column-based conflict detection, row-based conflict detection is used.

You have the following options for resolving conflicts:

- **Consolidated** First in wins: uploaded updates that conflict are rejected.
- **Remote** Last in wins: uploaded updates are always applied.

  Only use this option with column-based conflict detection. Otherwise, you get the same results and better performance by choosing no conflict detection.
- **Timestamp** The newest update wins. To use this option, you must create and maintain a timestamp column for the table. This timestamp column should record the last time that a row was changed. The

column should exist on both the consolidated and remote databases. To work, your remote and consolidated databases must use the same time zone (preferably UTC) and their clocks must be synchronized.

- **Custom** You write your own resolve_conflict scripts. You do this in the **Events** tab after the wizard completes.

  See "Resolving conflicts with resolve_conflict scripts" [*MobiLink - Server Administration*].

To customize conflict detection and resolution

1. In **Model** mode, open the **Mappings** tab.
2. In the **Table Mappings** pane, select a remote table.
3. In the **Cflt. Det** dropdown list, choose **None**, **Row-based**, or **Column-based**. If you chose **None**, you are done.
4. If you chose **Row-based** or **Column-based**, choose **Consolidated**, **Remote**, **Timestamp**, or **Custom** from the **Cflt. Res** dropdown list.
5. If you choose **Timestamp** conflict resolution, open the **Conflict Resolution** tab in the lower pane and enter the name of a timestamp column to use.
6. If you choose **Custom** conflict resolution, open the **Events** tab and write a resolve_conflict script for the table.

See also

- "Handling conflicts" [*MobiLink - Server Administration*]

Modifying scripts in a model

In MobiLink **Model** mode, open the **Events** tab to:

- View and modify the scripts that were generated by the **Create Synchronization Model Wizard**.
- Create new scripts.

The top of the **Events** tab tells you the group that the selected script belongs to. All scripts for a single table are grouped together for your convenience. The top of the **Events** tab also tells you the name of the selected script and whether it was generated by the **Create Synchronization Model Wizard**, whether it was user-defined, or whether a generated script was overridden. It also tells you whether the synchronization logic is written in SQL, .NET, or Java.

When you add or change a script, the script becomes fully under your control; it does not change automatically when you change a related setting in **Model** mode. For example, if you change a download_delete_cursor for a model and then clear **Del** in **Model** mode, your customized download_delete_cursor is not affected.

You can use options in the **File** menu to restore generated scripts you have changed or to remove new scripts you have added. Select the script(s) you want to restore or remove and choose **File** to see your options.

To find a script for a particular table, you can open the **Mappings** tab, select the row, and choose **File** » **Go To Events**. The **Events** tab opens at the appropriate table.

Authenticating to external servers in Model mode

To authenticate to external POP3, IMAP, or LDAP servers, open the **Authentication** tab in **Model** mode and select **Enable custom authentication for this synchronization model**.

You must enter information about the host and port, or for LDAP servers, the URL of the LDAP server.

For more information about these fields, see "External authenticator properties" [*MobiLink - Client Administration*].

Setting up server-initiated synchronization in Model mode

Server-initiated synchronization allows you to initiate synchronization on the client when something changes on the consolidated database. **Model** mode provides a way for you to set up server-initiated synchronization. This method provides a limited version of server-initiated synchronization that is easy to set up and run.

Notifications tab

To set up server-initiated synchronization (Sybase Central Model mode and Deploy Synchronization Model Wizard)

1. Use the **Create Synchronization Model Wizard** to create a MobiLink model.
2. With your model open in **Model** mode, open the **Notification** tab at the top of the model.
3. Select **Enable server-initiated synchronization**.
4. Select a consolidated database table to use for notification.

   A change to the data in this table results in a notification being sent to the remote database. The notification triggers a synchronization.

   You can only choose tables for this purpose for which you have defined a timestamp-based download cursor (the default). The notification is based on the contents of the download cursor.

   See "Writing download_cursor scripts" [*MobiLink - Server Administration*].
5. Select a polling interval. This is the time between polls. You can choose a predefined polling interval or you can enter an interval. The default is 30 seconds.

   If the Notifier loses the database connection, it recovers automatically at the first polling interval after the database becomes available again.
6. Optionally, change the isolation level of the Notifier's database connection. The default is read committed.

   Be aware of the consequences of setting the isolation level. Higher levels increase contention, and may adversely affect performance. Isolation level 0 allows reads of uncommitted data—data which may eventually be rolled back.

7. Optionally, change the gateway by which notifications are sent. The default is a default_device_tracker gateway. See "Gateways and carriers" [*MobiLink - Server-Initiated Synchronization*].

To deploy a model with server-initiated synchronization

1. Deploy your model:
   a. From the **File** menu, choose **Deploy**.
   b. Follow the instructions in the **Deploy Synchronization Model Wizard**.
      See "Deploying models" on page 43.
   c. On the **Server-Initiated Synchronization Listener** page, configure options for your Listener.
2. The model is deployed. For information about the files that are created, see "Synchronizing a deployed model" on page 45.
3. To use server-initiated synchronization, you must:
   a. Start the MobiLink server.
   b. Perform a first synchronization (if one has not yet been done).
   c. Start the Listener.
4. Navigate to the directory you chose when you first started the **Create Synchronization Model Wizard**. It holds your model with the extension *.mlsm*. It also holds the following sub-directories:
   - *\mlsrv*
   - *\remote*
   - *\consolidated*

About server-initiated synchronization (not in Model mode)

In the **Model** mode version of server-initiated synchronization, the MobiLink server uses a download_cursor script for a table to determine when to initiate a synchronization. It does this by using the download_cursor script to generate a request_cursor for the Notifier. When using this version of server-initiated synchronization, you cannot customize your request_cursor.

See "Writing download_cursor scripts" [*MobiLink - Server Administration*] and "request_cursor property" [*MobiLink - Server-Initiated Synchronization*].

Model mode also sets up a default device tracker gateway for sending notifications. You can customize your gateway. See "Gateways and carriers" [*MobiLink - Server-Initiated Synchronization*].

While Sybase Central **Model** mode provides a simplified version of server-initiated synchronization, you can also set up a complete version of server-initiated synchronization.

For a description of the complete implementation of server-initiated synchronization, see MobiLink - Server-Initiated Synchronization [*MobiLink - Server-Initiated Synchronization*].

For an overview of what is required to set up server-initiated synchronization outside of **Model** mode, see "Quick start to server-initiated synchronization" [*MobiLink - Server-Initiated Synchronization*].

Updating schemas in Model mode

The **Update Schema Wizard** allows you to update the consolidated and remote database schemas in your model.

The **Update Schema Wizard** is most useful after you have deployed your model and:

- You made a change to the remote database schema that needs to be included in the model.
- You made a change to the consolidated database schema that needs to be included in the model.

  For example, you need to run Update Schema before redeploying a model that created timestamp-based download for one or more tables. The previous deployment changed the schema of the consolidated database by adding a timestamp column or shadow table, so the schema needs to be updated.

Unlike the **Create New Remote Tables** window, which adds remote tables to the model, the **Update Schema Wizard** does not map the tables. You need to create table mappings using the **Mappings** tab. See "Modifying the remote database that your model creates" on page 33.

To update the schema

1. In **Model** mode, from the **File** menu, choose **Update Schema**.

   The **Update Schema Wizard** appears.
2. Choose the schema that you want to update.

   The **Update Schema Wizard** compares schemas in the model with the schemas of the databases.

   - **The Consolidated Database Schema** The consolidated schema in the model is updated. The remote schema in the model is unchanged.
   - **The Remote Database Schema** The remote schema in the model is updated. The consolidated schema in the model is unchanged.
   - **Both The Consolidated And Remote Database Schemas** Both the consolidated and remote schemas are updated in the model to match the schemas of the existing databases.
3. Follow the instructions in the wizard. Each page has online help.
4. Click **Finish**.

   When you click **Finish**, any connections to the consolidated and remote databases are closed. You are now working offline. No changes are made outside the model until you deploy the model: the consolidated database does not change and the remote database is not created or changed until that time.
5. Map the new remote tables in the **Mappings** tab. See "Modifying table and column mappings" on page 32.

Deploying models

When you are ready to try your model, you deploy it with the **Deploy Synchronization Model Wizard**. There are multiple things that can be deployed:

- Changes to the consolidated database.
- SQL Anywhere or UltraLite remote databases (you can choose to create a database, or add tables to an existing empty database, or use an existing database that already has your remote tables).
- Batch files to deploy the model (the generated batch files have variable declarations at the beginning that you can edit before running the batch files).
- Batch files to run the MobiLink server and the MobiLink client.
- Server-initiated synchronization configuration.

When you deploy a model, the model file is saved.

Deploying to the consolidated database

The **Deploy Synchronization Model Wizard** provides two options for deploying to the consolidated database:

- Apply your model directly to your consolidated database by populating MobiLink system tables and creating all required shadow tables, columns, triggers, and stored procedures. It also optionally creates batch files to run your MobiLink application.
- Create a SQL file that contains all the same changes. You can inspect this file, alter it, and run it anytime. The effect is identical to applying the changes directly.

To deploy a consolidated database from a SQL file

- When you ran the **Deploy Synchronization Model Wizard**, if you chose to create a file to run later (on the **Consolidated Database Deployment Destination** page), you must run the batch file that is located in the *consolidated* sub-folder of your model. This file creates all the objects you chose to have created in the consolidated database, including synchronization scripts, shadow tables, and triggers. It can also register MobiLink users in the consolidated database.

  To run this file, navigate to the *consolidated* directory and run the file that ends with *_consolidated.bat*. You must include connection information. For example, run:

  ```
  MyModel_consolidated.bat
  "dsn=my_odbc_datasource;uid=myuserid;pwd=mypassword"
  ```

  For some drivers, the DSN can have the userid and password so they do not need to be specified.

Note
If your deployment creates shadow tables, you must connect to the consolidated database as either the owner of the base tables for which shadow tables are created, or as an administrator.

Deploying remote databases

You can choose to use an existing remote database or have the wizard create one for you. The wizard can create remote databases directly or you can have it create a SQL file that you run to create remote databases.

The wizard creates a remote database (either SQL Anywhere or UltraLite) with default database creation options using the database owner that you specified in the model. Alternatively, you can create a remote database outside of the **Create Synchronization Model Wizard** with your own custom settings and use the wizard to add the required remote tables, or you can deploy to an existing remote database that already has the remote tables.

To deploy a remote database from a SQL file

- When you ran the **Deploy Synchronization Model Wizard**, if you chose to create a file to run later (on the **New SQL Anywhere Remote Database** page or **New UltraLite Remote Database** page), you must run the batch file in the *remote* directory. This file creates all the objects you chose to have created in the remote database, including tables, publications, subscriptions, and MobiLink users.

  To run this file, navigate to the *remote* directory and run the file that ends with *_remote.bat*. For example, run:

  ```
  MyModel_remote.bat
  ```

  You are prompted for a password if you are using an existing remote database.

Deploying batch files to run synchronization tools

The wizard can create the following batch files:

- A batch file to run the MobiLink server with options that you specify.
- For SQL Anywhere remote databases, a batch file to run dbmlsync with options that you specify.
- For UltraLite remote databases, a batch file to run ulsync with options that you specify. Ulsync is used for testing synchronization, so it helps you get started when you don't have a working UltraLite application.
- If you are setting up server-initiated synchronization, the **Deploy Synchronization Model Wizard** can also create batch files to run the Notifier and the Listener.

To deploy a model

1. In **Model** mode, choose **File** » **Deploy**.

   The **Deploy Synchronization Model Wizard** appears.
2. Follow the instructions in the wizard. Each page has online help.
3. When you are finished, the changes you selected are deployed. If there are existing files of the same name, they are overwritten.
4. To synchronize your application, see "Synchronizing a deployed model" on page 45.

Redeploying a model

After deploying a model, you can alter it. You can do this by making changes in **Model** mode and then redeploying. You can also alter your deployed application in Sybase Central **Admin** mode or by directly changing the database using system procedures or other methods. However, when you alter a deployed model outside of **Model** mode, you cannot reverse-engineer the changes back into the model. When you redeploy the model, changes to your synchronization application that were made outside the model are overwritten.

If you make any changes to the schema of the remote or consolidated databases, you need to update the schema in the model before you redeploy. Deployment often causes schema changes, so you may need to update the schema even if you haven't made any other changes. For example, if you deploy a model that adds a timestamp column to each synchronized table on the consolidated database (which is the default behavior when you create a model), you need to update the consolidated schema in the model before redeploying. Likewise, if you add a table to the consolidated database and then want to redeploy, you need to update the consolidated schema in the model and then create new remote tables.

To run the **Update Schema Wizard**, choose **File** » **Update Schema**.

See "Updating schemas in Model mode" on page 42.

Synchronizing a deployed model

When you deploy a model, directories and files are optionally created in the location you chose on the first page of the **Create Synchronization Model Wizard**. The files and directories are named according to the model name you chose at that time.

Assume you named your model **MyModel** and saved it under *c:\SyncModels*. Depending on the deployment options you chose, you might have the following files:

Directories (based on example name and location)	Description and contents (based on example name)
c:\SyncModels	Contains your model file, saved as *MyModel.mlsm*.
c:\SyncModels\MyModel	Contains folders holding your deployment files.
c:\SyncModels\MyModel\consolidated	Contains deployment files for the consolidated database: • *MyModel_consolidated.sql* - a SQL file for setting up the consolidated database. • *MyModel_consolidated.bat* - a batch file for running the SQL file.

Directories (based on example name and location)	Description and contents (based on example name)
c:\SyncModels\MyModel\mlsrv	Contains deployment files for the MobiLink server: • *MyModel_mlsrv.bat* - a batch file for running the MobiLink server. If you have set up server-initiated synchronization, it also starts the Notifier.
c:\SyncModels\MyModel\remote	Contains deployment files for the remote databases: • *dblsn.txt* - if you set up server-initiated synchronization, this is a text file with Listener option settings. It is used by *MyModel_dblsn.bat*. • *MyModel_dblsn.bat* - if you set up server-initiated synchronization, this is a batch file for running the Listener. • *MyModel_dbmlsync.bat* - if you deployed a SQL Anywhere remote database, this is a batch file for synchronizing SQL Anywhere databases with dbmlsync. • *MyModel_remote.bat* - a batch file for running *MyModel_remote.sql*. • *MyModel_remote.db* - if you chose to create a new SQL Anywhere remote database, this is the database. • *MyModel_remote.sql* - a SQL file for setting up the new SQL Anywhere remote database. • *MyModel_remote.udb* - if you chose to create a new UltraLite remote database, this is the database. • *MyModel_ulsync.bat* - if you deployed an UltraLite database, a batch file for testing synchronization with an UltraLite remote database using the ulsync utility.

Running the batch files

You must run the batch files that are created by the **Deploy Synchronization Model Wizard** from the command line, and you must include connection information. You may need to create ODBC data sources before running these batch files.

See "Working with ODBC data sources" [*SQL Anywhere Server - Database Administration*].

To synchronize your model using batch files

1. If you have not yet run MobiLink setup scripts on consolidated database, run them before deploying.

   See "Setting up a consolidated database" [*MobiLink - Server Administration*].

2. When you ran the **Deploy Synchronization Model Wizard**, if you chose to create a file to run later (on the **Consolidated Database Deployment Destination** page), you must run the batch file that is located in the *consolidated* sub-folder of your model. This file creates all the objects you chose to have created in the consolidated database, including synchronization scripts, shadow tables, and triggers. It can also register MobiLink users in the consolidated database.

To run this file, navigate to the *consolidated* directory and run the file that ends with *_consolidated.bat*. You must include connection information. For example, run:

```
MyModel_consolidated.bat "dsn=MY_ODBC_DATASOURCE"
```

3. When you ran the **Deploy Synchronization Model Wizard**, if you chose to create a file to run later (on the **New SQL Anywhere Remote Database** page or **New UltraLite Remote Database** page), you must run the batch file in the *remote* directory. This file creates all the objects you chose to have created in the remote database, including tables, publications, subscriptions, and MobiLink users.

   To run this file, navigate to the *remote* directory and run the file that ends with *_remote.bat*. For example, run:

   ```
   MyModel_remote.bat
   ```

   You are prompted for a password if you are using an existing remote database.

4. Start the MobiLink server by running *mlsrv\MyModel_mlsrv.bat*. If you set up server-initiated synchronization, this also starts the Notifier. You must include connection information. For example, run:

   ```
   MyModel_mlsrv.bat "dsn=MY_ODBC_DATASOURCE"
   ```

5. Synchronize.

   For a SQL Anywhere remote database:

   - Grant REMOTE DBA authority to a user other than DBA (recommended). For example, execute the following in Interactive SQL:

     ```
     GRANT REMOTE DBA
     TO userid, IDENTIFIED BY password
     ```

     Connect as the user with REMOTE DBA authority.
   - Start the remote database that is located in the *remote* directory. For example, run:

     ```
     dbeng11 MyModel_remote.db
     ```
   - Start dbmlsync, the SQL Anywhere MobiLink client. Run the file that ends with *_dbmlsync.bat* in the *remote* directory. You must include connection information. For example, run:

     ```
     MyModel_dbmlsync.bat "uid=dba;pwd=sql;eng=MyModel_remote"
     ```

   For an UltraLite remote database:

   - To test your synchronization, run the file that ends with *_ulsync.bat* in the *remote* directory.
   - Alternatively, run your UltraLite application.

6. If you set up server-initiated synchronization, you need to perform a first synchronization and then start the Listener. The first synchronization is required to create a remote ID file. To start the Listener, run the file that ends with *_dblsn.bat* in the *remote* directory. For example, run:

   ```
   MyModel_dblsn.bat
   ```

See also

- "GRANT REMOTE DBA statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*]

- “Permissions for dbmlsync” [*MobiLink - Client Administration*]

Part II. MobiLink Tutorials

This part provides tutorials that show you how to set up and use MobiLink technology. These range from very introductory tutorials for new users to demonstrations of how to use advanced features.

CHAPTER 3

Tutorial: Introduction to MobiLink

Contents

Introduction to MobiLink tutorial

This tutorial shows you how to develop a MobiLink application, including setting up the consolidated database, creating ODBC data sources and remote databases, and configuring database objects to fine-tune your application.

Required software

- SQL Anywhere 11

Competencies and experience

Basic computer skills.

Goals

You understand:

- the basic concepts of MobiLink
- how to develop a MobiLink application

Suggested background reading

For an introduction to MobiLink, see:

- "Introducing MobiLink synchronization" on page 3
- "MobiLink consolidated databases" [*MobiLink - Server Administration*]
- "MobiLink models" on page 25
- "Synchronization techniques" [*MobiLink - Server Administration*]

Lesson 1: Ensuring unique primary keys

This tutorial uses the sample database Demo, which has not been set up for MobiLink synchronization. Its tables all have primary keys, but they are unique only within the Demo database. Since Demo is about to become the consolidated database in a synchronization system, its primary keys must be unique within the entire system—including all remote databases.

Lesson 2: Run the Create Synchronization Model Wizard

This tutorial uses the demo.db sample database as your consolidated database. Your SQL Anywhere installation includes a MobiLink sample database called CustDB, but it is already set up as a MobiLink consolidated database. This tutorial starts from scratch and shows you how to set up a SQL Anywhere database to use as a consolidated database.

To run the Create Synchronization Model Wizard

1. Start the wizard.
   - Choose **Programs** » **SQL Anywhere 11** » **Sybase Central**.

     Sybase Central opens.
   - In the **Task** pane, choose **Create A Synchronization Model**. (If the **Task** pane is not open, click **View** » **Tasks**.)

     The **Create Synchronization Model Wizard** appears.
2. Type a name and location for your model.
   - For this tutorial, name your model **MLDemo**.
   - Save the model in *C:\temp*.
   - Click **Next**.
3. The **Primary Key Requirements** page appears. This page is informational only. Select all three checkboxes and click **Next**.
4. The **Consolidated Database Schema** page appears. In this page, you select the database you want to use as your consolidated database. For this tutorial, you use the sample database called Demo. This sample database has not been set up for MobiLink synchronization.
   - Click **Choose A Consolidated Database**.

     The **Connect** window appears.
   - Select **ODBC Data Source Name**, and click the down arrow.
   - From the dropdown list, select **SQL Anywhere 11 Demo**.
5. A window appears asking if you want to install System Setup. Click **Yes**.

   System objects are installed that make the demo database able to perform MobiLink synchronization.
6. The **Consolidated Database Schema** page appears again, showing that you are connected to SQL Anywhere 11 demo. Click **Next**.
7. The **Remote Database Schema** page appears. Select **No, Create a New Remote Database Schema** and click **Next**.
8. The **New Remote Database Schema** page appears. Click **Select All** and then click **Next**.
9. The **Download Type** page appears. Leave the default, **Timestamp-based Download**. Click **Next**.
10. The **Timestamp Download Options** page appears. Leave the default and click **Next**.
11. The **Download Deletes** page appears. Leave the defaults and click **Next**.

12. The **Download Subset** page appears. Leave the default and click **Next**.
13. The **Upload Conflict Detection** page appears. Choose **Row-based Conflict Detection** and click **Next**.
14. The **Upload Conflict Resolution** page appears. Choose **Consolidated Database Wins** and click **Next**.
15. The **Production, Script Version and Description** page appears. Click **Next**.
16. The **Completing The Create Synchronization Model** page appears. Click **Finish**.

    Model mode opens, showing your model with all the settings you chose in the wizard.

CHAPTER 4

Exploring the CustDB sample for MobiLink

Contents

Introduction to the MobiLink CustDB tutorial

CustDB is a sales-status application. The CustDB sample is a valuable resource for the MobiLink developer. It provides you with examples of how to implement many of the techniques you need to develop MobiLink applications.

The application has been designed to illustrate several common synchronization techniques. To get the most out of this chapter, study the sample application as you read.

A version of CustDB is supplied for each supported operating system and for each supported database type.

For the locations of CustDB and setup instructions, see "Setting up the CustDB consolidated database" on page 60.

Scenario

The CustDB scenario is as follows.

A consolidated database is located at the head office. The following data is stored in the consolidated database:

- The MobiLink system tables that hold the synchronization metadata, including the synchronization scripts that implement synchronization logic.
- The CustDB data, including all customer, product, and order information, stored in the rows of base tables.

There are two types of remote databases, mobile managers and sales representatives.

Each mobile sales representative's database contains all products but only those orders assigned to that sales representative, while a mobile manager's database contains all products and orders.

Synchronization design

The synchronization design in the CustDB sample application uses the following features:

- **Complete table downloads** All rows and columns of the ULProduct table are shared in their entirety with the remote databases.
- **Column subsets** All rows, but not all columns, of the ULCustomer table are shared with the remote databases.
- **Row subsets** Different remote users get different sets of rows from the ULOrder table.

  For more information about row subsets, see "Partitioning rows among remote databases" [*MobiLink - Server Administration*].
- **Timestamp-based synchronization** This is a way of identifying changes that were made to the consolidated database since the last time a device synchronized. The ULCustomer and ULOrder tables are synchronized using a method based on timestamps.

  See "Timestamp-based downloads" [*MobiLink - Server Administration*].
- **Snapshot synchronization** This is a simple method of synchronization that downloads all rows in every synchronization. The ULProduct table is synchronized in this way.

See "Snapshot synchronization" [*MobiLink - Server Administration*].

- **Primary key pools to maintain unique primary keys** It is essential to ensure that primary key values are unique across a complete MobiLink installation. The primary key pool method used in this application is one way of ensuring unique primary keys.

  See "Using primary key pools" [*MobiLink - Server Administration*].

  For other ways to ensure that primary keys are unique, see "Maintaining unique primary keys" [*MobiLink - Server Administration*].

For an entity-relationship diagram of the CustDB tables, see "About the CustDB sample database" [*SQL Anywhere 11 - Introduction*].

Further reading

- "Exploring the CustDB samples for UltraLite" [*UltraLite - Database Management and Reference*]

CustDB setup

This section describes the pieces that make up the code for the CustDB sample application and database. These include:

- The sample SQL scripts, located in the *samples-dir\MobiLink\CustDB*.
- The application code, located in *samples-dir\UltraLite\CustDB*.
- Platform-specific user interface code, located in subdirectories of *samples-dir\UltraLite\CustDB* named for each operating system.

Note
For more information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].

Setting up the CustDB consolidated database

The CustDB consolidated database can be any MobiLink supported consolidated database.

SQL Anywhere CustDB

A SQL Anywhere CustDB consolidated database is provided in *samples-dir\UltraLite\CustDB\custdb.db*. A DSN called SQL Anywhere 11 CustDB is included with your installation.

You can rebuild this database using the file *samples-dir\UltraLite\CustDB\newdb.bat*.

If you want to explore the way the CustDB sample is created, you can view the file *samples-dir\MobiLink\CustDB\syncsa.sql*.

CustDB for other RDBMSs

The following SQL scripts are provided in *samples-dir\MobiLink\CustDB* to build the CustDB consolidated database as any one of these supported RDBMSs:

RDBMS	Custdb setup script
Adaptive Server Enterprise	*custase.sql*
SQL Server	*custmss.sql*
Oracle	*custora.sql*
DB2 LUW	*custdb2.sql*
MySQL	*custmys.sql*

The following procedures create a CustDB consolidated database for each of the supported RDBMS.

For more information about preparing a database for use as a consolidated database, see "Setting up a consolidated database" [*MobiLink - Server Administration*].

To set up a consolidated database (Adaptive Server Enterprise, DB2 LUW, MySQL, Oracle, SQL Server)

1. Create a database in your RDBMS.
2. Add the MobiLink system tables by running one of the following SQL scripts, located in the *MobiLink\setup* subdirectory of your SQL Anywhere 11 installation:
   - For an Adaptive Server Enterprise consolidated database, run *syncase.sql*.
   - For a MySQL consolidated database, run *syncmys.sql*.
   - For an Oracle consolidated database, run *syncora.sql*.
   - For a SQL Server consolidated database, run *syncmss.sql*.
3. Add sample user tables to the CustDB database by running one of the following SQL scripts, located in *samples-dir\MobiLink\CustDB*:
   - For an Adaptive Server Enterprise consolidated database, run *custase.sql*.
   - For a MySQL consolidated database, run *custmys.sql*.
   - For an Oracle consolidated database, run *custora.sql*.
   - For a SQL Server consolidated database, run *custmss.sql*.
4. Create an ODBC data source called CustDB that references your database on the client machine.
   - Choose **Start** » **Programs** » **SQL Anywhere 11** » **ODBC Administrator**.
   - Click **Add**.
   - Select the appropriate driver from the list.

     Click **Finish**.
   - Name the ODBC data source CustDB.
   - Click the **Login** tab. Enter the **User ID** and **Password** for your database.

To set up a consolidated database (DB2 LUW)

1. Create a consolidated database on the DB2 LUW server. For the purposes of this tutorial, call it CustDB.
2. Ensure that the default table space (usually called USERSPACE1) uses 8 KB pages.

   If the default table space does not use 8 KB pages, complete the following steps:
   - Verify that at least one of your buffer pools has 8 KB pages. If not, create a buffer pool with 8 KB pages.
   - Create a new table space and temporary table space with 8 KB pages.

     For more information, consult your DB2 LUW documentation.
3. Add the MobiLink system tables to the DB2 LUW consolidated database using the file *MobiLink\setup\syncdb2.sql*:
   - Change the connect command at the top of the file *syncdb2.sql*. Replace *DB2Database* with the name of your database (or its alias). In this example, the database is called CustDB. You can also add your DB2 user name and password as follows:

```
connect to CustDB user userid using password ~
```

- Open a DB2 LUW Command Window on either the server or client computer. Run *syncdb2.sql* by typing the following command:

  ```
  db2 -c -ec -td~ +s -v -f syncdb2.sql
  ```

4. Copy *custdb2.class*, located in the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere installation, to the *SQLLIB\FUNCTION* directory on your DB2 LUW server. This class contains procedures used for the CustDB sample.
5. Add data tables to the CustDB database:
   - If necessary, change the connect command in custdb2.sql. For example, you could add the user name and password as follows. Replace *userid* and *password* with your user name and password.

     ```
     connect to CustDB user userid using password
     ```

   - Open a DB2 Command Window on either the server or client computer.
   - Run custdb2.sql by typing the following command:

     ```
     db2 -c -ec -td~ +s -v -f custdb2.sql
     ```

   - When processing is complete, enter the following command to close the command window:

     ```
     exit
     ```

6. Create an ODBC data source called CustDB that references the DB2 LUW database on the DB2 LUW client.
   - Start the ODBC Administrator:

     From the **Start** menu, choose **Programs** » **SQL Anywhere 11** » **ODBC Administrator**.

     The **ODBC Data Source Administrator** appears.
   - On the **User DSN** tab, click **Add**.

     The **Create New Data Source** window appears.
   - Select the ODBC driver for your DB2 LUW database. For example, choose IBM DB2 UDB ODBC Driver. Click **Finish**.

     For information about how to configure your ODBC driver, see:

     - Your DB2 LUW documentation
     - Recommended ODBC Drivers for MobiLink
7. Run the *custdb2setuplong* Java application on the DB2 LUW client as follows. This application resets the CustDB example in the database. After the initial setup, you can run this application at any time to reset the CustDB database by typing the same command line.
   - If you use a name other than CustDB for the data source, you must modify the connection code in *custdb2setuplong.java* and recompile it as follows. If the path specified by the system variable *%db2tempdir%* contains spaces, you must enclose the path in quotation marks.

     ```
     javac -g -classpath %db2tempdir%\java\jdk\lib\classes.zip;
     %db2tempdir%\java\db2java.zip;
     %db2tempdir%\java\runtime.zip custdb2setuplong.java
     ```

   - Type the following, where *userid* and *password* are the user name and password for connecting to the CustDB ODBC data source.


```
java custdb2setuplong userid password
```

See also

- "IBM DB2 LUW consolidated database" [*MobiLink - Server Administration*]

Setting up an UltraLite remote database

The following procedure creates a remote database for CustDB. The CustDB remote database must be an UltraLite database.

The application logic for the remote database is located in *samples-dir\UltraLite\CustDB*. It includes the following files:

- **Embedded SQL logic** The file *custdb.sqc* contains the SQL statements needed to query and modify information from the UltraLite database, as well as the calls required to start synchronization with the consolidated database.
- **C++ API logic** The file *custdbcomp.cpp* contains the C++ API logic.
- **User-interface features** These features are stored separately, in platform-specific subdirectories of *Samples\UltraLite\CustDB*.

You complete the following steps to install the sample application to a remote device that is running UltraLite:

To install the sample application to a remote device

1. Start the consolidated database.
2. Start the MobiLink server.
3. Install the sample application to your remote device.
4. Start the sample application on the remote device.
5. Synchronize the sample application.

Example

The following example installs the CustDB sample on a Palm device running against a DB2 consolidated database.

1. Ensure that the consolidated database is running:

   For a DB2 LUW database, open a DB2 Command Window. Type the following command, where *userid* and *password* are the user ID and password for connecting to the DB2 LUW database:

   ```
   db2 connect to CustDB user userid using password
   ```

2. Start the MobiLink server:

   For a DB2 LUW database, at a command prompt, run the following command:

   ```
   mlsrv11 -c "DSN=CustDB" -zp
   ```

3. Install the sample application to your Palm device:
   - On your PC, start Palm Desktop.

- Click **Quick Install** on the Palm Desktop toolbar.
- Click **Add**. Browse to *custdb.prc* in the *UltraLite\palm\68k* subdirectory of your SQL Anywhere 11 installation.
- Click **Open**.
- HotSync your Palm device.

4. Start the CustDB sample application on your Palm device:
   - Place your Palm device in its cradle.

     When you start the sample application for the first time, you are prompted to synchronize to download an initial copy of the data. This step is required only the first time you start the application. After that, the downloaded data is stored in the UltraLite database.
   - Launch the sample application.

     From the **Applications** view, tap CustDB.

     An initial window appears, prompting you for an employee ID.
   - Enter an employee ID.

     For the purpose of this tutorial, enter a value of 50. The sample application also allows values of 51, 52, or 53, but behaves slightly differently in these cases.

     For more information about the behavior of each user ID, see "Users in the CustDB sample" on page 68.

     A window tells you that you must synchronize before proceeding.
   - Synchronize your application.

     Use HotSync to obtain an initial copy of the data.
   - Confirm that the data has been synchronized into the application.

     From the **Applications** view, tap the CustDB application. The display shows an entry sheet for a customer, with entries.
5. Synchronize the remote application with the consolidated database. You only need to complete this step when you have made changes to the database.
   - Ensure that the consolidated database and the MobiLink server are running.
   - Place the Palm device in its cradle.
   - Press the HotSync button to synchronize.

Tables in the CustDB databases

The table definitions for the CustDB database are in platform-specific files in *samples-dir\MobiLink\CustDB*. (For information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].)

For an entity-relationship diagram of the CustDB tables, see "About the CustDB sample database" [*SQL Anywhere 11 - Introduction*].

Both the consolidated and the remote databases contain the following five tables, although their definitions are slightly different in each location.

ULCustomer

The ULCustomer table contains a list of customers.

In the remote database, ULCustomer has the following columns:

- **cust_id** A primary key column that holds a unique integer that identifies the customer.
- **cust_name** A 30-character string containing the name of the customer.

In the consolidated database, ULCustomer has the following additional column:

- **last_modified** A timestamp containing the last time the row was modified. This column is used for timestamp-based synchronization.

ULProduct

The ULProduct table contains a list of products.

In both the remote and consolidated databases, ULProduct has the following columns:

- **prod_id** A primary key column that contains a unique integer that identifies the product.
- **price** An integer identifying the unit price.
- **prod_name** A 30-character string that contains the name of the product.

ULOrder

The ULOrder table contains a list of orders, including details of the customer who placed the order, the employee who took the order, and the product being ordered.

In the remote database, ULOrder has the following columns:

- **order_id** A primary key column that holds a unique integer identifying the order.
- **cust_id** A foreign key column referencing ULCustomer.
- **prod_id** A foreign key column referencing ULProduct.
- **emp_id** A foreign key column referencing ULEmployee.
- **disc** An integer containing the discount applied to the order.
- **quant** An integer containing the number of products ordered.
- **notes** A 50-character string containing notes about the order.

- **status** A 20-character string describing the status of the order.

In the consolidated database, ULOrder has the following additional column:

- **last_modified** A timestamp containing the last time the row was modified. This column is used for timestamp-based synchronization.

ULOrderIDPool

The ULOrderIDPool table is a primary key pool for ULOrder.

In the remote database, ULOrderIDPool has the following column:

- **pool_order_id** A primary key column that holds a unique integer identifying the order ID.

In the consolidated database, ULOrderIDPool has the following additional columns:

- **pool_emp_id** An integer column containing the employee ID of the owner of the remote database to which the order ID has been assigned.
- **last_modified** A timestamp containing the last time the row was modified.

ULCustomerIDPool

The ULCustomerIDPool table is a primary key pool for ULCustomer.

In the remote database, ULCustomerIDPool has the following column:

- **pool_cust_id** A primary key column that holds a unique integer identifying the customer ID.

In the consolidated database, ULCustomerIDPool has the following additional columns:

- **pool_emp_id** An integer column containing the employee ID that is used for a new employee generated at a remote database.
- **last_modified** A timestamp containing the last time the row was modified.

The following tables are contained in the consolidated database only:

ULIdentifyEmployee_nosync

The ULIdentifyEmployee_nosync table exists only in the consolidated database. It has a single column as follows:

- **emp_id** This primary key column contains an integer representing an employee ID.

ULEmployee

The ULEmployee table exists only in the consolidated database. It contains a list of sales employees.

ULEmployee has the following columns:

- **emp_id** A primary key column that holds a unique integer identifying the employee.
- **emp_name** A 30-character string containing the name of the employee.

ULEmpCust

The ULEmpCust table controls which customers' orders are downloaded. If the employee needs a new customer's orders, inserting the employee ID and customer ID forces the orders for that customer to be downloaded.

- **emp_id** A foreign key to ULEmployee.emp_id.
- **cust_id** A foreign key to ULCustomer.cust_id. The primary key consists of emp_id and cust_id.
- **action** A character used to determine if an employee record should be deleted from the remote database. If the employee no longer requires a customer's orders, set to D (delete). If the orders are still required, the action should be set to null.

  A logical delete must be used in this case so that the consolidated database can identify which rows to remove from the ULOrder table. Once the deletes have been downloaded, all records for that employee with an action of D can also be removed from the consolidated database.
- **last_modified** A timestamp containing the last time the row was modified. This column is used for timestamp-based synchronization.

ULOldOrder and ULNewOrder

These tables exists only in the consolidated database. They are for conflict resolution and contain the same columns as ULOrder. In SQL Anywhere and Microsoft SQL Server, these are temporary tables. In Adaptive Server Enterprise, these are normal tables and @@spid. DB2 LUW and Oracle do not have temporary tables, so MobiLink needs to be able to identify which rows belong to the synchronizing user. Since these are base tables, if five users are synchronizing, they might each have a row in these tables at the same time.

For more information about @@spid, see "Variables" [*SQL Anywhere Server - SQL Reference*].

Users in the CustDB sample

There are two types of users in the CustDB sample, sales people and mobile managers. The differences are as follows:

- **Sales people** User IDs 51, 52, and 53 identify remote databases that are associated with sales people. Sales people can perform the following tasks:
  - View lists of customers and products.
  - Add new customers.
  - Add or delete orders.
  - Scroll through the list of outstanding orders.
  - Synchronize changes with the consolidated database.
- **Mobile managers** User ID 50 identifies the remote database associated with the mobile manager. The mobile manager can perform the same tasks as a sales person. In addition, the mobile manager can perform the following task:
  - Accept or deny orders.

Synchronizing CustDB

The following sections describe the CustDB sample's synchronization logic.

Synchronization logic source code

You can use Sybase Central to inspect the synchronization scripts in the consolidated database.

Script types and events

The *custdb.sql* file adds each synchronization script to the consolidated database by calling ml_add_connection_script or ml_add_table_script.

Example

The following lines in *custdb.sql* add a table-level script for the ULProduct table, which is executed during the download_cursor event. The script consists of a single SELECT statement.

```
call ml_add_table_script(
'CustDB 10.0',
'ULProduct', 'download_cursor',
'SELECT prod_id, price, prod_name FROM ULProduct' )
go
```

Synchronizing orders in the CustDB sample

Business rules

The business rules for the ULOrder table are as follows:

- Orders are downloaded only if they are not approved or the status is null.
- Orders can be modified at both the consolidated and remote databases.
- Each remote database contains only the orders assigned to an employee.

Downloads

Orders can be inserted, deleted, or updated at the consolidated database. The scripts corresponding to these operations are as follows:

- **download_cursor** The first parameter in the download_cursor script is the last download timestamp. It is used to ensure that only rows that have been modified on either the remote or the consolidated database since the last synchronization are downloaded. The second parameter is the employee ID. It is used to determine which rows to download.

  The download_cursor script for CustDB is as follows:

  ```
  CALL ULOrderDownload( {ml s.last_table_download}, {ml s.username} )
  ```

  The ULOrderDownload procedure for CustDB is as follows:

  ```
  CREATE PROCEDURE ULOrderDownload ( IN LastDownload timestamp, IN
  EmployeeID integer )
  ```

```
BEGIN`
  SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant,
o.notes, o.status
    FROM ULOrder o, ULEmpCust ec
    WHERE o.cust_id = ec.cust_id
    AND ec.emp_id = EmployeeID
    AND ( o.last_modified >= LastDownload
    OR ec.last_modified >= LastDownload)
    AND ( o.status IS NULL  OR  o.status != 'Approved' )
    AND ( ec.action IS NULL )
END
```

- **download_delete_cursor** The download_delete_cursor script for CustDB is as follows:

```
SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant,
o.notes, o.status
    FROM ULOrder o, dba.ULEmpCust ec
  WHERE o.cust_id = ec.cust_id
    AND ( ( o.status = ''Approved'' AND o.last_modified >= {ml
s.last_table_download} )
    OR ( ec.action = ''D''  )  )
    AND ec.emp_id = {ml s.username}
```

Uploads

Orders can be inserted, deleted or updated at the remote database. The scripts corresponding to these operations are as follows:

- **upload_insert** The upload_insert script for CustDB is as follows:

```
INSERT INTO ULOrder ( order_id, cust_id, prod_id, emp_id, disc, quant,
notes, status )
   VALUES( {ml r.order_id, r.cust_id, r.prod_id, r.emp_id, r.disc, r.quant,
r.notes, r.status } )
```

- **upload_update** The upload_update script for CustDB is as follows:

```
UPDATE ULOrder
  SET cust_id = {ml r.cust_id},
      prod_id = {ml r.prod_id},
      emp_id = {ml r.emp_id},
      disc = {ml r.disc},
      quant = {ml r.quant},
      notes = {ml r.notes},
      status = {ml r.status}
    WHERE order_id = {ml r.order_id}
```

- **upload_delete** The upload_delete script for CustDB is as follows:

```
DELETE FROM ULOrder WHERE order_id = {ml r.order_id}
```

- **upload_fetch** The upload_fetch script for CustDB is as follows:

```
SELECT order_id, cust_id, prod_id, emp_id, disc, quant, notes, status
   FROM ULOrder WHERE order_id = {ml r.order_id}
```

- **upload_old_row_insert** The upload_old_row_insert script for CustDB is as follows:

```
INSERT INTO ULOldOrder ( order_id, cust_id, prod_id, emp_id, disc, quant,
notes, status )
   VALUES( {ml r.order_id, r.cust_id, r.prod_id, r.emp_id, r.disc, r.quant,
r.notes, r.status } )
```

- **upload_new_row_insert** The upload_new_row_insert script for CustDB is as follows:


```
INSERT INTO ULNewOrder ( order_id, cust_id, prod_id, emp_id, disc, quant,
notes, status )
   VALUES( {ml r.order_id, r.cust_id, r.prod_id, r.emp_id, r.disc, r.quant,
r.notes, r.status } )
```

Conflict resolution

- **resolve_conflict** The resolve_conflict script for CustDB is as follows:

  ```
  CALL ULResolveOrderConflict
  ```

  The ULResolveOrderConflict procedure for CustDB is as follows:

  ```
  CREATE PROCEDURE ULResolveOrderConflict()
  BEGIN
    -- approval overrides denial
    IF 'Approved' = (SELECT status FROM ULNewOrder) THEN
      UPDATE ULOrder o
      SET o.status = n.status, o.notes = n.notes
      FROM ULNewOrder n
      WHERE o.order_id = n.order_id;
    END IF;
    DELETE FROM ULOldOrder;
    DELETE FROM ULNewOrder;
  END
  ```

Synchronizing customers in the CustDB sample

Business rules

The business rules governing customers are as follows:

- Customer information can be modified at both the consolidated and remote databases.
- Both the remote and consolidated databases contain a complete listing of customers.

Downloads

Customer information can be inserted or updated at the consolidated database. The script corresponding to these operations is as follows:

- **download_cursor** The following download_cursor script downloads all customers for whom information has changed since the last time the user downloaded information.

  ```
  SELECT cust_id, cust_name FROM ULCustomer WHERE last_modified >= {ml
  s.last_table_download}
  ```

Uploads

Customer information can be inserted, updated, or deleted at the remote database. The scripts corresponding to these operations are as follows:

- **upload_insert** The upload_insert script for CustDB is as follows:

  ```
  INSERT INTO ULCustomer( cust_id, cust_name )
     VALUES( {ml r.cust_id, r.cust_name } )
  ```

- **upload_update** The upload_update script for CustDB is as follows:

```
UPDATE ULCustomer SET cust_name = {ml r.cust_name}
   WHERE cust_id = {ml r.cust_id}
```

Conflict detection is not performed on this table.

- **upload_delete** The upload_delete script for CustDB is as follows:

```
DELETE FROM ULCustomer WHERE cust_id = {ml r.cust_id}
```

Synchronizing products in the CustDB sample

Business rules

All rows are downloaded for ULProduct—this is called snapshot synchronization.

See "Snapshot synchronization" [*MobiLink - Server Administration*].

The business rules for the ULProduct table are as follows:

- Products can only be modified at the consolidated database.
- Each remote database contains all of the products.

Downloads

Product information can be inserted, deleted, or updated at the consolidated database. The script corresponding to these operations is as follows:

- **download_cursor** The following download_cursor script downloads all of the rows and columns of the ULProduct table at each synchronization:

```
SELECT prod_id, price, prod_name FROM ULProduct
```


Maintaining the customer and order primary key pools

The CustDB sample database uses primary key pools in order to maintain unique primary keys in the ULCustomer and ULOrder tables. The primary key pools are the ULCustomerIDPool and ULOrderIDPool tables.

ULCustomerIDPool

The following scripts are defined in the ULCustomerIDPool table:

Downloads

- **download_cursor**

```
SELECT pool_cust_id FROM ULCustomerIDPool
   WHERE last_modified >= {ml s.last_table_download}
     AND pool_emp_id = {ml s.username}
```

Uploads

- **upload_insert** The upload_insert script for CustDB is as follows:

```
INSERT INTO ULCustomerIDPool ( pool_cust_id )
   VALUES( {ml r.pool_cust_id} )
```

- **upload_delete** The upload_delete script for CustDB is as follows:

```
DELETE FROM ULCustomerIDPool
   WHERE pool_cust_id = {ml r.pool_cust_id}
```

- **end_upload** The following end_upload script ensures that after each upload 20 customer IDs remain in the customer ID pool:

```
CALL ULOrderIDPool_maintain( {ml s.username} )
```

  The UL_CustomerIDPool_maintain procedure for CustDB is as follows:

```
CREATE PROCEDURE ULCustomerIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
  DECLARE pool_count INTEGER;
  -- Determine how many ids to add to the pool
  SELECT COUNT(*) INTO pool_count
    FROM ULCustomerIDPool
    WHERE pool_emp_id = syncuser_id;
  -- Top up the pool with new ids
  WHILE pool_count < 20 LOOP
    INSERT INTO ULCustomerIDPool ( pool_emp_id )
      VALUES ( syncuser_id );
    SET pool_count = pool_count + 1;
  END LOOP;
END
```


ULOrderIDPool

The following scripts are defined in the ULOrderIDPool table:

Downloads

- **download_cursor** The download_cursor script for CustDB is as follows:

```
SELECT pool_order_id FROM ULOrderIDPool
  WHERE last_modified >= {ml s.last_table_download}
    AND pool_emp_id = {ml s.username}
```

Uploads

- **end_upload** The following end_upload script ensures that after each upload 20 order IDs remain in the order ID pool.

```
CALL ULOrderIDPool_maintain( {ml s.username} )
```

  The UL_OrderIDPool_maintain procedure for CustDB is as follows:

```
ALTER PROCEDURE ULOrderIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
  DECLARE pool_count INTEGER;
  -- Determine how many ids to add to the pool
  SELECT COUNT(*) INTO pool_count
    FROM ULOrderIDPool
    WHERE pool_emp_id = syncuser_id;
  -- Top up the pool with new ids
  WHILE pool_count < 20 LOOP
    INSERT INTO ULOrderIDPool ( pool_emp_id )
      VALUES ( syncuser_id );
    SET pool_count = pool_count + 1;
  END LOOP;
END
```

- **upload_insert** The upload_insert script for CustDB is as follows:

```
INSERT INTO ULOrderIDPool ( pool_order_id )
   VALUES( {ml r.pool_order_id}
```

- **upload_delete** The upload_delete script for CustDB is as follows:

```
DELETE FROM ULOrderIDPool
   WHERE pool_order_id = {ml r.pool_order_id}
```


Cleanup

To restart the sample, reset the data in the CustDB database.

To reset the data in the CustDB database

1. Install the ULDBUtil on your device:
   - For a Palm device, start Palm Desktop on your PC.
   - Click **Install** on the Palm Desktop toolbar.
   - Click **Add**. Browse to *uldbutil.prc* in the *UltraLite\palm\68k* subdirectory of your SQL Anywhere 11 installation.
   - Click **Done**.
   - HotSync your Palm device.
2. Delete the data using ULDBUtil:
   - For a Palm device, tap the ULDBUtil icon.
   - Select CustDB and tap **Delete Data**.
   - HotSync your Palm device.

Further reading

For more information about script types, see “Script types” [*MobiLink - Server Administration*].

For reference material, including detailed information about each script and its parameters, see “Synchronization events” [*MobiLink - Server Administration*].

CHAPTER 5

Exploring the MobiLink Contact sample

Contents

Introduction to the Contact sample tutorial

The Contact sample is a valuable resource for the MobiLink developer. It provides you with an example of how to implement many of the techniques you need to develop MobiLink applications.

The Contact sample application includes a SQL Anywhere consolidated database and two SQL Anywhere remote databases. It illustrates several common synchronization techniques. To get the most out of this chapter, study the sample application as you read.

Although the consolidated database is a SQL Anywhere database, the synchronization scripts consist of SQL statements that should work with minimal changes on other database management systems.

The Contact sample is in *samples-dir**MobiLink**Contact*. For an overview, see the readme in the same location. (For information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].)

Synchronization design

The synchronization design in the Contact sample application uses the following features:

- **Column subsets** A subset of the columns of the Customer, Product, SalesRep, and Contact tables on the consolidated database are shared with the remote databases.
- **Row subsets** All of the columns but only one of the rows of the SalesRep table on the consolidated database are shared with each remote database. See "Partitioning rows among remote databases" [*MobiLink - Server Administration*].
- **Timestamp-based synchronization** This is a way of identifying changes that were made to the consolidated database since the last time a device synchronized. The Customer, Contact, and Product tables are synchronized using a method based on timestamps. See "Timestamp-based downloads" [*MobiLink - Server Administration*].

Contact sample setup

A Windows batch file called *build.bat* is provided to build the Contact sample databases. On UNIX systems, the file is *build.sh*. You may want to examine the contents of the batch file. It performs the following actions:

- Creates ODBC data source definitions for a consolidated database and each of two remote databases.
- Creates a consolidated database named *consol.db* and loads the MobiLink system tables, database schema, some data, synchronization scripts, and MobiLink user names into the database.
- Creates two remote databases, each named *remote.db*, in subdirectories named *remote_1* and *remote_2*. Loads information common to both databases and applies customizations. These customizations include a global database identifier, a MobiLink user name, and subscriptions to two publications.

To build the Contact sample

1. At a command prompt, navigate to *samples-dir\MobiLink\Contact*.
2. Run *build.bat* (Windows) or *build.sh* (Unix).

For information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].

Running the Contact sample

The Contact sample includes batch files that perform initial synchronizations and illustrate MobiLink server and dbmlsync command lines. You can examine the contents of the following batch files, located in *samples-dir\MobiLink\Contact*, in a text editor:

- *step1.bat*
- *step2.bat*
- *step3.bat*

To run the Contact sample

1. Start the MobiLink server.
   - At a command prompt, navigate to *samples-dir\MobiLink\Contact* and run the following command:

     ```
     step1
     ```

     This command runs a batch file that starts the MobiLink server in a verbose mode. This mode is useful during development or troubleshooting, but has a significant performance impact and so would not be used in a routine production environment.
2. Synchronize both remote databases.
   - At a command prompt, navigate to *samples-dir\MobiLink\Contact*.
   - Run the following command:

     ```
     step2
     ```

This is a batch file that synchronizes both remote databases.

3. Shut down the MobiLink server.
   - At a command prompt, navigate to *samples-dir\MobiLink\Contact*.
   - Run the following command:

     ```
     step3
     ```

     This is a batch file that shuts down the MobiLink server.

To explore how synchronization works in the Contact sample, you can use Interactive SQL to modify the data in the remote and consolidated databases, and use the batch files to synchronize.

Tables in the Contact databases

The table definitions for the Contact database are located in the following files, all under your samples directory:

- *MobiLink\Contact\build_consol.sql*
- *MobiLink\Contact\build_remote.sql*

Both the consolidated and the remote databases contain the following three tables, although their definition is slightly different in each place.

SalesRep

Each sales representative occupies one row in the SalesRep table. Each remote database belongs to a single sales representative.

In each remote database, SalesRep has the following columns:

- **rep_id** A primary key column that contains an identifying number for the sales representative.
- **name** The name of the representative.

In the consolidated database only, there is also an ml_username column holding the MobiLink user name for the representative.

Customer

This table holds one row for each customer. Each customer is a company with which a single sales representative does business. There is a one-to-many relationship between the SalesRep and Customer tables.

In each remote database, Customer has the following columns:

- **cust_id** A primary key column holding an identifying number for the customer.
- **name** The customer name. This is a company name.
- **rep_id** A foreign key column that references the SalesRep table. Identifies the sales representative assigned to the customer.

In the consolidated database, there are two additional columns, last_modified and active:

- **last_modified** The last time the row was modified. This column is used for timestamp-based synchronization.
- **active** A BIT column that indicates if the customer is currently active (1) or if the company no longer deals with this customer (0). If the column is marked inactive (0) all rows corresponding to this customer are deleted from remote databases.

Contact

This table holds one row for each contact. A contact is a person who works at a customer company. There is a one-to-many relationship between the Customer and Contact tables.

In each remote database, Contact has the following columns:

- **contact_id** A primary key column holding an identifying number for the contact.

- **name** The name of the individual contact.
- **cust_id** The identifier of the customer for whom the contact works.

In the consolidated database, the table also has the following columns:

- **last_modified** The last time the row was modified. This column is used for timestamp-based synchronization.
- **active** A BIT column that indicates if the contact is currently active (1) or if the company no longer deals with this contact (0). If the column is marked inactive (0) the row corresponding to this contact is deleted from remote databases.

Product

Each product sold by the company occupies one row in the Product table. The Product table is held in a separate publication so that remote databases can synchronize the table separately.

In each remote database, Product has the following columns:

- **id** A primary key column that contains a number to identify the product.
- **name** The name of the item.
- **size** The size of the item.
- **quantity** The number of items in stock. When a sales representative takes an order, this column is updated.
- **unit_price** The price per unit of the product.

In the consolidated database, the Product table has the following additional columns:

- **supplier** The company that manufactures the product.
- **last_modified** The last time the row was modified. This column is used for timestamp-based synchronization.
- **active** A BIT column that indicates if the product is currently active (1). If the column is marked inactive (0), the row corresponding to this product is deleted from remote databases.

In addition to these tables, a set of tables is created at the consolidated database only. These include the product_conflict table, which is a temporary table used during conflict resolution, and a set of tables for monitoring MobiLink activities owned by a user named mlmaint. Scripts to create the MobiLink monitoring tables are in the file *samples-dir\MobiLink\Contact\mlmaint.sql*.

Users in the Contact sample

Several different database user IDs and MobiLink user names are included in the Contact sample.

Database user IDs

The two remote databases belong to sales representatives Samuel Singer (rep_id 856) and Pamela Savarino (rep_id 949).

When connecting to their remote database, these users both use the default SQL Anywhere user ID **dba** and password **SQL**.

Each remote database also has a user ID **sync_user** with password **sync_user**. This user ID is employed only on the dbmlsync command line. It is a user with REMOTE DBA authority, and so can perform any operation when connected from dbmlsync, but has no authority when connected from any other application. The widespread availability of the user ID and password is thus not a problem.

At the consolidated database, there is a user named **mlmaint**, who owns the tables used for monitoring MobiLink synchronization statistics and errors. This user has no right to connect. The assignment of the tables to a separate user ID is done simply to separate the objects from the others in the schema for easier administration in Sybase Central and other utilities.

MobiLink user names

MobiLink user names are distinct from database user IDs. Each remote device has a MobiLink user name in addition to the user ID they use when connecting to a database. The MobiLink user name for Samuel Singer is SSinger. The MobiLink user name for Pamela Savarino is PSavarino. The MobiLink user name is stored or used in the following locations:

- At the remote database, the MobiLink user name is added using a CREATE SYNCHRONIZATION USER statement.
- At the consolidated database, the MobiLink user name and password are added using the mluser utility.
- During synchronization, the MobiLink password for the connecting user is supplied on the dbmlsync command line listed in *MobiLink\Contact\step2.bat*.
- The MobiLink server supplies the MobiLink user name as a parameter to many of the scripts during synchronization.
- The SalesRep table at the consolidated database has an ml_username column. The synchronization scripts match the MobiLink user name parameter against the value in this column.

Synchronizing the Contact sample

The following sections describe the Contact sample's synchronization logic.

Synchronizing sales representatives in the Contact sample

The synchronization scripts for the SalesRep table illustrates **snapshot synchronization**. Regardless of whether a sales representative's information has changed, it is downloaded.

See "Snapshot synchronization" [*MobiLink - Server Administration*].

Business rules

The business rules for the SalesRep table are as follows:

- The table must not be modified at the remote database.
- A sales representative's MobiLink user name and rep_id value must not change.
- Each remote database contains a single row from the SalesRep table, corresponding to the remote database owner's MobiLink user name.

Downloads

- **download_cursor** At each remote database, the SalesRep table contains a single row. There is very little overhead for the download of a single row, so a simple snapshot download_cursor script is used:

  ```
  SELECT rep_id, name
  FROM SalesRep
  WHERE ? IS NOT NULL
  AND ml_username = ?
  ```

  The first parameter in the script is the last download timestamp, which is not used. The IS NOT NULL expression is a dummy expression supplied to use the parameter. The second parameter is the MobiLink user name.

Uploads

This table should not be updated at the remote database, so there are no upload scripts for the table.

Synchronizing customers in the Contact sample

The synchronization scripts for the Customer table illustrate **timestamp-based synchronization** and partitioning rows. Both of these techniques minimize the amount of data that is transferred during synchronization while maintaining consistent table data.

See:

- "Timestamp-based downloads" [*MobiLink - Server Administration*]
- "Partitioning rows among remote databases" [*MobiLink - Server Administration*]

Business rules

The business rules governing customers are as follows:

- Customer information can be modified at both the consolidated and remote databases.
- Periodically, customers may be reassigned among sales representatives. This process is commonly called territory realignment.
- Each remote database contains only the customers they are assigned to.

Downloads

- **download_cursor** The following download_cursor script downloads only active customers for whom information has changed since the last successful download. It also filters customers by sales representative.

```
SELECT cust_id, Customer.name, Customer.rep_id
FROM Customer key join SalesRep
WHERE Customer.last_modified >= ?
AND SalesRep.ml_username = ?
AND Customer.active = 1
```

- **download_delete_cursor** The following download_delete_cursor script downloads only customers for whom information has changed since the last successful download. It deletes all customers marked as inactive or who are not assigned to the sales representative.

```
SELECT cust_id
FROM Customer key join SalesRep
WHERE Customer.last_modified >= ?
AND ( SalesRep.ml_username != ? OR Customer.active = 0 )
```

  If rows are deleted from the Customer table at the consolidated database, they do not appear in this result set and so are not deleted from remote databases. Instead, customers are marked as inactive.

  When territories are realigned, this script deletes those customers no longer assigned to the sales representative. It also deletes customers who are transferred to other sales representatives. Such additional deletes are flagged with a SQLCODE of 100 but do not interfere with synchronization. A more complex script could be developed to identify only those customers transferred away from the current sales representative.

  The MobiLink client performs cascading deletes at the remote database, so this script also deletes all contacts who work for customers assigned to some other sales representative.

Uploads

Customer information can be inserted, updated, or deleted at the remote database. The scripts corresponding to these operations are as follows:

- **upload_insert** The following upload_insert script adds a row to the Customer table, marking the customer as active:

```
INSERT INTO Customer(
 cust_id, name, rep_id, active )
VALUES ( ?, ?, ?, 1 )
```

- **upload_update** The following upload_update script modifies the customer information at the consolidated database. Conflict detection is not done on this table.

```
UPDATE Customer
SET name = ?, rep_id = ?
WHERE cust_id = ?
```

- **upload_delete** The following upload_delete script marks the customer as inactive at the consolidated database. It does not delete a row.

```
UPDATE Customer
SET active = 0
WHERE cust_id = ?
```

Synchronizing contacts in the Contact sample

The Contact table contains the name of a person working at a customer company, a foreign key to the customer, and a unique integer identifying the contact. It also contains a last_modified timestamp and a marker to indicate whether the contact is active.

Business rules

The business rules for this table are as follows:

- Contact information can be modified at both the consolidated and remote databases.
- Each remote database contains only those contacts who work for customers they are assigned to.
- When customers are reassigned among sales representatives, contacts must also be reassigned

Trigger

A trigger on the Customer table is used to ensure that the contacts get picked up when information about a customer is changed. The trigger explicitly alters the last_modified column of each contact whenever the corresponding customer is altered:

```
CREATE TRIGGER UpdateCustomerForContact
AFTER UPDATE OF rep_id ORDER 1
ON DBA.Customer
REFERENCING OLD AS old_cust NEW as new_cust
FOR EACH ROW
BEGIN
  UPDATE Contact
  SET Contact.last_modified = new_cust.last_modified
  FROM Contact
  WHERE Contact.cust_id = new_cust.cust_id
END
```

By updating all contact records whenever a customer is modified, the trigger ties the customer and their associated contacts together. Whenever a customer is modified, all associated contacts are modified too, and the customer and associated contacts are downloaded together on the next synchronization.

Downloads

- **download_cursor** The download_cursor script for Contact is as follows:

```
SELECT contact_id, contact.name, contact.cust_id
FROM ( contact JOIN customer ) JOIN salesrep
ON contact.cust_id = customer.cust_id
 AND customer.rep_id = salesrep.rep_id
WHERE Contact.last_modified >= ?
```



```
AND salesrep.ml_username = ?
AND Contact.active = 1
```

This script retrieves all contacts that are active, that have been changed since the last time the sales representative downloaded (either explicitly or by modification of the corresponding customer), and that are assigned to the representative. A join with the Customer and SalesRep table is needed to identify the contacts associated with this representative.

- **download_delete_cursor** The download_delete_cursor script for Contact is as follows:

```
SELECT contact_id
FROM ( Contact JOIN Customer ) JOIN SalesRep
ON Contact.cust_id = Customer.cust_id
 AND Customer.rep_id = SalesRep.rep_id
WHERE Contact.last_modified >= ?
 AND Contact.active = 0
```

The automatic use of cascading referential integrity by the MobiLink client deletes contacts when the corresponding customer is deleted from the remote database. The download_delete_cursor script therefore has to delete only those contacts marked as inactive.

Uploads

Contact information can be inserted, updated, or deleted at the remote database. The scripts corresponding to these operations are as follows:

- **upload_insert** The following upload_insert script adds a row to the Contact table, marking the contact as active:

```
INSERT INTO Contact (
 contact_id, name, cust_id, active )
VALUES ( ?, ?, ?, 1 )
```

- **upload_update** The following upload_update script modifies the contact information at the consolidated database:

```
UPDATE Contact
SET name = ?, cust_id = ?
WHERE contact_id = ?
```

Conflict detection is not done on this table.

- **upload_delete** The following upload_delete script marks the contact as inactive at the consolidated database. It does not delete a row.

```
UPDATE Contact
SET active = 0
WHERE contact_id = ?
```

Synchronizing products in the Contact sample

The scripts for the Product table illustrate conflict detection and resolution.

The Product table is kept in a separate publication from the other tables so that it can be downloaded separately. For example, if the price changes and the sales representative is synchronizing over a slow link, they can download the product changes without uploading their own customer and contact changes.

Business rules

The only change that can be made at the remote database is to change the quantity column, when an order is taken.

Downloads

- **download_cursor** The following download_cursor script downloads all rows changed since the last time the remote database synchronized:

```
SELECT id, name, size, quantity, unit_price
FROM product
WHERE last_modified >= ?
AND active = 1
```

- **download_delete_cursor** The following download_delete_cursor script removes all products no longer sold by the company. These products are marked as inactive in the consolidated database.

```
SELECT id, name, size, quantity, unit_price
FROM product
WHERE last_modified >= ?
AND active = 0
```

Uploads

Only UPDATE operations are uploaded from the remote database. The major feature of these upload scripts is a conflict detection and resolution procedure.

If two sales representatives take orders and then synchronize, each order is subtracted from the quantity column of the Product table. For example, if Samuel Singer takes an order for 20 baseball hats (product ID 400), he changes the quantity from 90 to 70. If Pamela Savarino takes an order for 10 baseball hats before receiving this change, she changes the column in her database from 90 to 80.

When Samuel Singer synchronizes his changes, the quantity column in the consolidated database is changed from 90 to 70. When Pamela Savarino synchronizes her changes, the correct action is to set the value to 60. This setting is accomplished by detecting the conflict.

The conflict detection scheme includes the following scripts:

- **upload_update** The following upload_update script is a straightforward UPDATE at the consolidated database:

```
UPDATE product
SET name = ?, size = ?, quantity = ?, unit_price = ?
WHERE product.id = ?
```

- **upload_fetch** The following upload_fetch script fetches a single row from the Product table for comparison with the old values of the uploaded row. If the two rows differ, a conflict is detected.

```
SELECT id, name, size, quantity, unit_price
FROM Product
WHERE id = ?
```

- **upload_old_row_insert** If a conflict is detected, the old values are placed into the product_conflict table for use by the resolve_conflict script. The row is added with a value of O (for Old) in the row_type column.


```
INSERT INTO DBA.product_conflict(
 id, name, size, quantity, unit_price, row_type )
VALUES( ?, ?, ?, ?, ?, 'O' )' )
```

- **upload_new_row_insert** The following script adds the new values of the uploaded row into the product_conflict table for use by the resolve_conflict script:

```
INSERT INTO DBA.product_conflict(
 id, name, size, quantity, unit_price, row_type )
VALUES( ?, ?, ?, ?, ?, 'N' )
```

Conflict resolution

- **resolve_conflict** The following script resolves the conflict by adding the difference between new and old rows to the quantity value in the consolidated database:

```
UPDATE Product
SET p.quantity = p.quantity
              - old_row.quantity
              + new_row.quantity
FROM Product p,
     DBA.product_conflict old_row,
     DBA.product_conflict new_row
WHERE p.id = old_row.id
    AND p.id = new_row.id
    AND old_row.row_type = 'O'
    AND new_row.row_type = 'N'
```

Monitoring statistics and errors in the Contact sample

The Contact sample contains some simple error reporting and monitoring scripts. The SQL statements to create these scripts are in the file *MobiLink\Contact\mlmaint.sql*.

The scripts insert rows into tables created to hold the values. For convenience, the tables are owned by a distinct user, mlmaint.

CHAPTER 6

Tutorial: Using MobiLink with an Oracle 10g consolidated database

Contents

Introduction to Oracle tutorial

This tutorial shows you to mobilize an Oracle 10g database using MobiLink. It sets up synchronization between an Oracle 10g consolidated database and a SQL Anywhere remote database. You could also set up an UltraLite remote database.

The purpose of this tutorial is to mobilize the data pertaining to a sales team. In this scenario, each salesperson is a remote synchronization client. Each salesperson has a local SQL Anywhere database that is synchronized to a corporate Oracle database at headquarters using MobiLink. Each salesperson access corporate data with their laptop or handheld device, and manipulates data from the remote database.

Required software

This tutorial assumes that you have a complete install of SQL Anywhere (including MobiLink) on your local computer where you are running the tutorial. This tutorial was created using Oracle Database 10g Release 2. The tutorial may work for other versions of Oracle, but this is not guaranteed. This tutorial also assumes that you have installed Oracle Database 10g on your local computer, but you may also access it remotely.

This tutorial assumes you performed a basic installation of Oracle Database 10g, which creates a starter database named orcl. The orcl database has the Order Entry (OE) and Human Relations (HR) sample schemas. Alternatively, you may manually install the sample schemas or use the Oracle Database Configuration Assistant. For more information about installing both sample schemas, see http://www.oracle.com/technology/obe/obe1013jdev/common/OBEConnection.htm.

This tutorial assumes that you can connect to Oracle as the SYS user with SYSDBA privileges. This is a requirement in Lesson 7 when you grant permission for the V_$TRANSACTION Oracle system view. The password for the SYS user is set during installation of an Oracle database.

Overview

This tutorial shows you how to:

- Make important considerations, such as synchronization directions for remote tables, when designing a remote schema.
- Add unique primary keys to consolidated and remote databases.
- Create an ODBC data source that connects MobiLink to an Oracle 10g database.
- Set up synchronization between a consolidated database and remote database using the **Create Synchronization Model Wizard**.
- Customize a synchronization model using Model mode.
- Deploy a consolidated database and remote database using the **Deploy Synchronization Model Wizard**.
- Synchronize the remote client with the consolidated database.

Suggested background reading

For an overview of MobiLink synchronization, see “Introducing MobiLink synchronization” on page 3.

For an overview of MobiLink models, see “MobiLink models” on page 25.

Lesson 1: Design the schemas

This tutorial assumes that you have installed the Order Entry (OE) and Human Relations (HR) sample schemas. The OE schema is used as the consolidated database. It encapsulates information about employees, orders, customers, and products. For this tutorial, you are primarily interested in the OE schema. However, you must refer to the EMPLOYEES table in the HR schema to get information about each individual salesperson. Here is a brief description of the relevant tables in the OE schema:

Table	Description
CUSTOMERS	Customers whose information is kept on record.
INVENTORIES	How much of each product is stored in each warehouse.
ORDER_ITEMS	List of products included in each order.
ORDERS	Record of a sale between a salesperson and a customer on a specific date.
PRODUCT_DESCRIPTION	Descriptions of each product in different languages.
PRODUCT_INFORMATION	A record of each product in the system.

Designing the remote schema

The first step is to design a remote schema. It is unnecessary and inefficient for each salesperson to have a copy of the entire consolidated database. The remote schema is designed so that it only contains information relevant to one particular salesperson. To achieve this, the remote schema is designed in the following way:

Consolidated table	Remote table
CUSTOMERS	Includes all rows.
INVENTORIES	Not included on remote.
ORDER_ITEMS	Filter by sales_rep_id.
ORDERS	Includes all rows.
PRODUCT_DESCRIPTION	Not included on remote.
PRODUCT_INFORMATION	Includes all rows.

Each salesperson needs to keep records of all customers and products, so that any product can be sold to any customer. This tutorial assumes that a salesperson always speaks the same language as the customer, so you do not need the PRODUCT_DESCRIPTION table. Each salesperson needs information about orders, but not orders related to other salespeople. This is achieved by filtering rows based on salesperson identifier.

The next step is to choose the synchronization direction of each table. You should consider what information a remote database needs to read and what information a remote database needs to create, change, or remove.

In this example, a specific salesperson needs a list of products and customers, but never entered a new product into the system. You are making the restriction that products and customers always enter the system from the consolidated database at headquarters. However, a salesperson needs to be able to record new orders on a regular basis. These factors lead to the following decisions about the synchronization in each table:

Table	Synchronization
CUSTOMERS	Download to remote only.
ORDER_ITEMS	Download and upload.
ORDER	Download and upload.
PRODUCT_INFORMATION	Download to remote only.

Lesson 2: Prepare the consolidated database

This tutorial assumes that you have installed the Order Entry (OE) sample database. Information about installing the sample schema can be found in the Oracle documentation or can be viewed online at http://www.oracle.com/technology/obe/obe1013jdev/common/OBEConnection.htm.

The OE database needs to be altered for use with MobiLink. Columns are dropped because they were created as user-defined types. You could translate these user-defined types into types that SQL Anywhere recognizes, but doing so is not relevant to this tutorial. Next, you must grant permission to the OE user to create triggers because MobiLink needs to create a few triggers using OE's credentials.

To prepare the consolidated database

1. Connect as the SYS user with SYSDBA privileges using the Oracle SQL Plus application. At a command prompt, run the following command:

   ```
   sqlplus SYS/your password for sys as SYSDBA
   ```

2. To drop columns created as user-defined types, run the following commands:

   ```
   ALTER TABLE OE.CUSTOMERS DROP COLUMN CUST_ADDRESS;
   ALTER TABLE OE.CUSTOMERS DROP COLUMN PHONE_NUMBERS;
   ALTER TABLE OE.CUSTOMERS DROP COLUMN CUST_GEO_LOCATION;
   ALTER TABLE OE.PRODUCT_INFORMATION DROP COLUMN WARRANTY_PERIOD;
   ```

3. Unlock the OE user and set the password to sql. Run the following command:

   ```
   ALTER USER OE IDENTIFIED BY sql ACCOUNT UNLOCK;
   ```

4. To grant permission to the OE user to create triggers, run the following command:

   ```
   GRANT CREATE ANY TRIGGER TO OE;
   ```

5. To drop the orders_customer foreign key and create a new foreign key that references the customer_id in the customers table, run the following commands:

   ```
   ALTER TABLE OE.ORDERS DROP CONSTRAINT ORDERS_CUSTOMER_ID_FK;
   ALTER TABLE OE.ORDERS ADD CONSTRAINT ORDERS_CUSTOMER_ID_FK
     FOREIGN KEY (CUSTOMER_ID) REFERENCES OE.CUSTOMERS (CUSTOMER_ID);
   ```

Adding unique primary keys

In a synchronization system, the primary key of a table is the only way to uniquely identify a row in different databases and the only way to detect conflicts. Every table that is being mobilized must have a primary key. The primary key must never be updated. You must also guarantee that a primary key value inserted at one database is not inserted in another database.

There are several ways of generating unique primary keys. For simplicity, the method of composite primary keys is used in this tutorial. This method creates primary keys with multiple columns that are unique across consolidated and remote databases.

To add unique primary keys to the consolidated database

1. Connect as the SYS user with SYSDBA privileges using the Oracle SQL Plus application. At a command prompt, run the following command:

   ```
   sqlplus SYS/your password for sys as SYSDBA
   ```

2. Values added to the SALES_REP_ID must be in the HR.EMPLOYEES table. The ORDERS_SALES_REP_FK foreign key enforces this rule. For simplicity, drop this foreign key by running the following command:

```
ALTER TABLE OE.ORDERS
DROP CONSTRAINT ORDERS_SALES_REP_FK;
```

3. The SALES_REP_ID column cannot be added as a primary key because it contains null values. For the purposes of this tutorial, replace the null values with a value of 1. Run the following command:

```
UPDATE OE.ORDERS
SET SALES_REP_ID = 1
WHERE SALES_REP_ID IS NULL;
```

4. The ORDER_ID column is the current primary key of the ORDERS table. To drop the current primary key, run the following command:

```
ALTER TABLE OE.ORDERS
DROP PRIMARY KEY CASCADE;
```

5. The composite primary key consists of the SALES_REP_ID column and the ORDER_ID column. To add the composite primary key, run the following command:

```
ALTER TABLE OE.ORDERS
ADD CONSTRAINT salesrep_order_pk PRIMARY KEY (sales_rep_id, order_id);
```

After running these commands, the MobiLink server should have no trouble connecting to the consolidated database and setting it up for synchronization with any number of remotes.

Adding foreign keys

Unique primary keys across all databases

In Lesson 4, the remote schema is created from the consolidated schema. This means that the remote schema has the same primary keys as the consolidated schema.

Columns were specifically chosen to ensure unique primary keys for all databases. For the ORDERS table, the primary key consists of the SALES_REP_ID and ORDER_ID columns. Any inserted value into the remote sales table must have an unique order number (the SALES_REP_ID value is always the same). This ensures uniqueness in each remote ORDERS table. The primary key in the consolidated ORDERS table prevents conflicts if multiple salespeople upload data. Each upload from a salesperson is unique to another salesperson because their SALES_REP_ID values are different.

Further reading

For more information about compliant RDBMSs, see "Introduction to consolidated databases" [*MobiLink - Server Administration*].

For more information about setting up Oracle as a consolidated database, see "Oracle consolidated database" [*MobiLink - Server Administration*].

For more information about different ways of generating unique primary keys, see "Maintaining unique primary keys" [*MobiLink - Server Administration*].

Lesson 3: Connect with MobiLink

In this lesson, you create an ODBC data source that connects MobiLink to the consolidated database.

To connect MobiLink to the consolidated database

1. Create an ODBC data source.

   You should use the iAnywhere driver for Oracle that comes with SQL Anywhere 11. Use the following configuration settings:

ODBC tab fields	Values
Data source name	oracle_cons
User ID	OE
Password	sql
SID	orcl
Procedure returns results	not selected.
Array size	60000

   This tutorial assumes you performed a basic installation of Oracle Database 10g, which creates a starter database named orcl. The Order Entry (OE) schema is automatically installed on orcl. If you installed the OE schema on another database, use the name of the database as the SID value.

2. To test the ODBC connection, click **Test Connection**.

   A **Connection Successful** window appears.

After configuring your ODBC data source, you can use the MobiLink plug-in to connect to the Oracle database and create a synchronization model.

Further reading

For more information about recommended ODBC drivers for MobiLink, see Recommended ODBC Drivers for MobiLink.

For more information about configuration options for the iAnywhere Solutions Oracle driver, see "iAnywhere Solutions Oracle driver" [*MobiLink - Server Administration*].

Lesson 4: Create the synchronization model

The **Create Synchronization Model Wizard** takes you through the steps of setting up synchronization between the consolidated database and remote database.

To create a synchronization model

1. Start Sybase Central.
2. From the **Tools** menu, choose **MobiLink 11** » **Set Up MobiLink Synchronization**.

   The **Create Synchronization Model Wizard** appears.
3. Type **sync_oracle** as the name and type the location of your new model.
4. On the **Primary Key Requirements** page, select all three checkboxes. (You guarantee unique primary keys in Lesson 2.)
5. On the **Consolidated Database Schema** page, connect to the consolidated database:
   a. Click **Choose The Consolidated Database**.

      The **Connect To Consolidated Database** window appears.
   b. Select **ODBC Data Source Name** and choose **oracle_cons**. Click **OK**.
   c. If this is the first time the consolidated database has been used by MobiLink, a window appears. It asks if you want to install the MobiLink system setup. Installing the MobiLink system setup adds MobiLink system tables and procedures. Click **Yes**.

      If you see an error message, ensure that the data source is configured properly.
   d. Once the MobiLink system tables and procedures are added to the consolidated database, the name, user, product and version appear on the page. Click **Next**.

      The **Loading Consolidated Database Schema** window appears. The tables from the consolidated database schemas are loaded.
6. Create a remote schema:
   a. On the **Remote Database Schema** page, select **No**, **Create A New Remote Database Schema** and click **Next**.
   b. On the **New Remote Database Schema** page, select the checkboxes for the following tables to include them in the remote schema and then click **Next**.
      - CUSTOMERS
      - ORDERS
      - ORDER_ITEMS
      - PRODUCT_INFORMATION
7. Choose and configure the download type:
   a. On the **Download Type** page, select **Timestamp-based Download**.

      Choosing timestamp-based downloads minimizes the amount of data that is transferred because only data that has been updated since the last download is transmitted.
   b. On the **Timestamp Download Options** page, select **Use Shadow Tables To Hold Timestamp Columns**.

Using shadow tables is often preferred because it does not require any changes to existing tables.

8. On the **Download Deletes** page, specify how to propagate record deletions to remote devices:
   a. To indicate that you want the remote database to download deletes, select **Yes**.
   b. Select **Use Shadow Tables To Record Deletions**.

      MobiLink creates shadow tables on the consolidated database to implement deletions.
9. On the **Download Subset** page, specify that the remote database should download only a subset of data from the consolidated database:
   a. Select **Yes**, **Download The Same Data To Each Remote**. (In Step 2 of the Model mode lesson, you specify how to download specific data to a remote database by using custom logic.)
10. On the **Upload Conflict Detection** page, select **No Conflict Detection**.

    Many applications require conflict detection, but this tutorial uses no conflict detection.
11. On the **Publication, Script Version and Description** page, type **sync_oracle_publication** as your publication and type **sync_oracle_scriptversion** as your script version.

    The publication is the object on the remote database that specifies what data is synchronized. MobiLink server scripts define how uploaded data from remotes should be applied to the consolidated database, and script versions group scripts. You can use different script versions for different applications, allowing you to maintain a single MobiLink server while synchronizing different applications.
12. Click **Finish**.

    Your model appears in Model mode.

Model mode

1. To specify the direction data is synchronized, set the directions in the **Mapping Direction** column as follows:
   - ORDERS and ORDER_ITEMS are bi-directional (both upload and download).
   - The remaining tables are download only.
2. Filter the rows downloaded to the remote database by remote ID:
   a. For the ORDERS table, change the **Download Subset** to **Custom**.
   b. Open the **Download Subset** tab at the bottom of the screen.
   c. The remote ID uniquely identifies a remote database. To filter rows by remote ID, add a restriction to the WHERE clause of the download_cursor script. To do this, type the following search condition in the **SQL Expression To Use In The Download Cursor's WHERE Clause** text box:

      ```
      "OE"."ORDERS"."SALES_REP_ID" = {ml s.remote_id}
      ```

      The download cursor script specifies what columns and rows are downloaded from each table to the remote database. The search condition ensures that you only download information about one salesperson, namely, the representative that has an identifier that equals the remote ID for the database.
3. Save the synchronization model.

   The synchronization model is complete and ready to be deployed.

Further reading

- "Setting up a consolidated database" [*MobiLink - Server Administration*]
- "MobiLink server system tables" [*MobiLink - Server Administration*]
- "MobiLink system procedures" [*MobiLink - Server Administration*]
- "Writing download_delete_cursor scripts" [*MobiLink - Server Administration*]
- "Handling conflicts" [*MobiLink - Server Administration*]
- "Resolving conflicts" [*MobiLink - Server Administration*]
- "Publishing data" [*MobiLink - Client Administration*]
- "Model mode" on page 32
- "Modifying the download type" on page 34
- "Modifying conflict detection and resolution" on page 38
- "Modifying table and column mappings" on page 32

Lesson 5: Deploy the synchronization model

The **Deploy Synchronization Model Wizard** allows you to deploy the consolidated database and remote database. You can deploy each of these individually or you can deploy both of them. The **Deploy Synchronization Model Wizard** takes you through the steps of configuring options for deployment.

To deploy the synchronization model

1. In the left pane of Sybase Central, right-click the synchronization model you just created and choose **Deploy** from the popup menu.

   The **Deploy Synchronization Model Wizard** appears.

2. Choose deployment settings:
   a. Select **Specify The Deployment Details For One Or More Of The Following**.
   b. Select **Consolidated Database**, **Remote Database and Synchronization Client**, and **MobiLink Server**.

      This instructs MobiLink to deploy each piece of the synchronization environment.

3. On the **Consolidated Database Deployment Destination** page, configure the options for deploying the consolidated database:
   a. Select **Save Changes To The Following SQL File**. For this tutorial, use the default location.

      MobiLink generates a *.sql* file that makes changes to the consolidated database to set up for synchronization.

   b. You may select **Connect To The Consolidated Database To Directly Apply The Changes**. This option causes MobiLink to apply the changes immediately to the consolidated database.

      You may also examine the *.sql* file later and make your own changes. Then, you must run the *.sql* file yourself.

4. On the same page, connect to the consolidated database:
   a. Click **Choose A Consolidated Database**.

      The **Connect To Consolidated Database** window appears.

   b. Select **ODBC Data Source Name** and choose **oracle_cons**. Click **OK**.

      The **Checking For MobiLink System Setup** window appears. Once the tables from the consolidated database have loaded, the name, user, product and version appear on the page.

   c. Click **Next**.
   d. A window appears and it asks if you want to create the *consolidated* directory. Click **Yes**.

      The **Loading Consolidated Database Schema** window appears. The tables from the consolidated database schemas are loaded.

5. On the **Remote Database Deployment** page, specify the type of remote database being deployed. Select **New SQL Anywhere Database**.
6. On the **New SQL Anywhere Remote Database** page, configure the options for creating a new remote database:
   a. Select **Make A Command File And A SQL File With Commands To Create The Database**.

MobiLink generates another *.sql* file with the commands to set up the remote database with all schema and synchronization information.

b. Use the default location of the SQL file.

c. Select **Create A Remote SQL Anywhere Database**.

If you choose not to do this, you must generate a new remote database, and then run the *.sql* file against it. This allows you to examine the *.sql* file later and make your own changes.

d. Use the default location of the remote SQL Anywhere database.

e. Click **Next**.

A window appears and it asks if you want to create the *remote* directory. Click **Yes**.

7. Configure the options for deploying the MobiLink server:

a. On the **MobiLink User** page, type **oracle_remote** as the MobiLink user name and type **oracle_pass** as the password.

The MobiLink user is stored in a MobiLink system table and is used to authenticate the user during synchronization.

b. On the **Synchronization Stream Parameters** page, choose TCP/IP as the communication protocol and enter 2439 as the port to use.

c. On the **Client Stream Parameters** page, type **localhost** as the host name of the MobiLink server computer.

You can optionally type your computer's name or IP address, the name or IP address of another network server you want to use, or other client stream options.

d. On the **MobiLink Server Stream Parameters** page, you can specify additional server stream options.

e. On the **Verbosity For MobiLink Server** page, choose the amount of log verbosity and use the default file name for the log file.

You can use the log file for troubleshooting if you encounter any errors.

f. On the **Options For MobiLink Server** page, type **oracle_mlsrv** as the name of the MobiLink server and use the default location of the command file.

You use the command file to start the MobiLink server.

g. Click **Next**.

A window appears and it asks if you want to create the *mlsrv* directory. Click **Yes**.

8. Configure the options for deploying the remote synchronization client:

a. On the **Verbosity For SQL Anywhere Remote Synchronization Client** page, choose the verbosity.

b. On the same page, use the default file name of the remote database log file.

c. On the **Advanced Options For SQL Anywhere Remote Synchronization Client** page, you can specify advanced options.

d. On the same page, use the default location of the command file and click **Next**.

MobiLink generates a command file that you can use to start synchronization.

9. Click **Finish**.

A **Deploying** window appears. When the deployment completes, it shows Status: Completed. Click **Close**.

Your consolidated database is fully configured for synchronization with many remote clients, and you have successfully deployed one remote client. If you want to deploy other remote clients, you can run this wizard again, making sure to create a new MobiLink user and opt out of deploying the consolidated database and MobiLink server. Since these have already been deployed, all you need to do is deploy other remote synchronization clients.

Further reading

- "Deploying models" on page 43
- "Creating a remote database" [*MobiLink - Client Administration*]
- "Introduction to MobiLink users" [*MobiLink - Client Administration*]

Lesson 6: Start the server and client

In this lesson, you start the MobiLink server and remote database.

Previously, you modified the download cursor script to download information related to one salesperson. In this lesson, you specify the salesperson by setting the remote ID to the salesperson identifier.

Start the MobiLink server

By default, MobiLink uses the snapshot/READ COMMITTED isolation level for upload and download. For the MobiLink server to make the most effective use of snapshot isolation, the Oracle account used by the MobiLink server must have access to the V_$TRANSACTION Oracle system view. If access is not given, a warning is issued and rows may be missed on download.

To grant access to the OE user

1. Connect as the SYS user with SYSDBA privileges using the Oracle SQL Plus application. At a command prompt, run the following command:

   ```
   sqlplus SYS/your password for sys as SYSDBA
   ```

2. To grant access to the V_$TRANSACTION Oracle system view, run the following command:

   ```
   GRANT SELECT ON SYS.V_$TRANSACTION TO OE;
   ```

To start the MobiLink server

1. At a command prompt, navigate to the directory where you created the synchronization model. (This is the root directory you chose in the first step of the **Create Synchronization Model Wizard**.)

   If you used the suggested directory names, you should have the following directories located in the root directory: *sync_oracle\mlsrv*.

2. Run the following command from the mlsrv directory:

   ```
   sync_oracle_mlsrv.bat "DSN=oracle_cons;UID=OE;PWD=sql;"
   ```

   - **sync_oracle_mlsrv.bat** is the command file created to start the MobiLink server.
   - **DSN** is your ODBC data source name.
   - **UID** is the user name you use to connect to the consolidated database.
   - **PWD** is the password you use to connect to the consolidated database.

   A **MobiLink Server Messages** window appears. When this command runs successfully, it shows `MobiLink Server Started`.

   If the MobiLink server fails to start, check your connection information for your consolidated database.

To start the remote database

1. At a command prompt, navigate to the directory where the **Deploy Synchronization Model Wizard** created your remote database.

   If you used the suggested directory names, you should have the following directories located in the root directory: *sync_oracle\remote*.

2. Start your remote SQL Anywhere database with the following command:

```
dbeng11 -n remote_eng sync_oracle_remote.db -n remote_db
```

- **dbeng11** is the database server used to start the SQL Anywhere database.
- **remote_eng** is the database server name.
- **sync_oracle_remote.db** is the database file that is started on remote_eng.
- **remote_db** is the name of the database on remote_eng.

A **SQL Anywhere Server Messages** window appears. When this command runs successfully, a SQL Anywhere database server named remote_eng starts and loads the database called remote_db.

Set the remote ID

In the remote schema, each remote database represents one salesperson. The synchronization scripts you wrote included logic that instructed the MobiLink server to download a subset of data based on the remote ID of the remote database. You must set the database's remote ID to the value of a valid salesperson identifier.

It is important to complete this step before the first synchronization because when the remote device synchronizes for the first time, it downloads all information related to the chosen salesperson.

To set the remote ID to a valid salesperson identifier

1. Choose a valid salesperson identifier:
   a. Connect as the SYS user with SYSDBA privileges using the Oracle SQL Plus application. At a command prompt, run the following command:

```
sqlplus SYS/your password for sys as SYSDBA
```

   b. To view a list of valid salesperson identifiers in the ORDERS table, execute the following statement:

```
SELECT COUNT( SALES_REP_ID ), SALES_REP_ID
FROM OE.ORDERS GROUP BY SALES_REP_ID;
```

   In this example, the remote database represents a salesperson with a SALES_REP_ID of 154.
   c. To exit Oracle, run the following command:

```
exit
```

2. To set the database's remote id to a value of 154, run the following command:

```
dbisql
-c "ENG=remote_eng;DBN=remote_db;UID=DBA;PWD=sql"
"SET OPTION PUBLIC.ml_remote_id='154'"
```

- **dbisql** is the application used to execute SQL commands against a SQL Anywhere database.
- **ENG** specifies the database server name remote_eng.
- **DBN** specifies the database name to remote_db.
- **UID** is the user name used to connect to your remote database.
- **PWD** is the password used to connect to your remote database.
- **SET OPTION PUBLIC.ml_remote_id='154'** is the SQL command used to set the remote ID to a value of 154.

Further reading

- "The SQL Anywhere database server" [*SQL Anywhere Server - Database Administration*]
- "Synchronizing a deployed model" on page 45
- "Running the MobiLink server" [*MobiLink - Server Administration*]
- "Remote IDs" [*MobiLink - Client Administration*]

Lesson 7: Synchronize

Now you are ready to synchronize the remote client for the first time. This is done with the MobiLink client program dbmlsync. Dbmlsync connects to the remote database, authenticates itself with the MobiLink server, and performs all of the uploads and downloads necessary to synchronize the remote and consolidated databases based on a publication in the remote database.

To synchronize the remote client

- At a command prompt, run the following command:

```
dbmlsync
-c "ENG=remote_eng;DBN=remote_db;UID=DBA;PWD=sql;"
-n sync_oracle_publication
-u oracle_remote -mp oracle_pass
```

  - **dbmlsync** is the synchronization application.
  - **ENG** specifies the name of the remote database server.
  - **DBN** specifies the name of the remote database.
  - **UID** is the user name used to connect to the remote database.
  - **PWD** is the password used to connect to the remote database.
  - **sync_oracle_publication** is the publication on the remote device that is used to perform the synchronization. (This publication was created by the **Create Synchronization Model Wizard**.)
  - **oracle_remote** is the user name used to authenticate with the MobiLink server.
  - **oracle_pass** is the password used to authenticate with the MobiLink server.

A SQL Anywhere MobiLink client messages window appears. The window shows you the progress. When this command runs successfully, the dbmlsync application populates the remote database with a subset of information from the consolidated database.

If synchronization fails, you start by checking the connection information you pass to the dbmlsync application, as well as the MobiLink user name and password. Failing that, check the publication name you used, and ensure that the consolidated database and MobiLink server are running. You can also examine the contents of the synchronization logs (server and client).

> **Note**
> If you are running the dbmlsync application on a different computer from your MobiLink server, you must pass in arguments that specify the location of the MobiLink server.

View the data

After successfully synchronizing the remote client to the consolidated database through the MobiLink server, the remote data should be populated with information relevant to one salesperson. You can verify this in Sybase Central using the SQL Anywhere 11 plug-in.

To view the data in the remote database

1. Start Sybase Central

2. Connect to the remote database:
   a. On the left pane, right-click **SQL Anywhere 11** and choose **Connect**.

      A **Connect** window appears.
   b. Type **DBA** as the **User Id** and **sql** as the **Password**.
   c. Under the **Identification** tab, type **remote_eng** as the **Server Name** and **remote_db** as the **Database File**.
   d. Click **OK**.
3. Choose the ORDERS table and then click the **Data** tab on the right pane.

   In the ORDERS tables, all the records are for the salesperson with an identifier of 154. This particular salesperson is not concerned with the sales information of other salespeople. For this reason, you set the synchronization scripts to filter out rows by the remote ID, and you set this database's remote ID to the value of a particular salesperson identifier. Now this particular salesperson's database takes up less space, and requires less time to synchronize. Since the remote database size is kept to a minimum, frequently performed operations such as entering a new sale or processing a refund on a previous sale runs faster and more efficiently.

Further reading

- "The synchronization process" on page 16
- "dbmlsync syntax" [*MobiLink - Client Administration*]

Cleanup

Regenerate the Order Entry database and remove all tutorial materials from your computer.

To regenerate the Order Entry database

- Run *oe_main.sql* to remove the current OE schema and install a new OE schema. The *oe_main.sql* file is found in *$ORACLE_HOME/demo/schema/order_entry*.

Alternatively, you may use the Oracle Database Configuration Assistant to create a new database with newly installed sample schemas.

To delete the synchronization model

1. Start Sybase Central.
2. Double-click **MobiLink 11** in the right pane.

   The **sync_oracle** model appears in the right pane.
3. Right click **sync_oracle** and choose **Delete**.

   The **Confirm Delete** window appears. Choose **Delete**.

To erase the remote database

- To erase the remote database, use the dberase utility. Run the following command:

```
dberase sync_oracle\remote\sync_oracle_remote.db
```


Further reading

For more information about running the MobiLink server, see "MobiLink server" [*MobiLink - Server Administration*].

For more information about writing synchronization scripts, see "Writing synchronization scripts" [*MobiLink - Server Administration*] and "Synchronization events" [*MobiLink - Server Administration*].

For an introduction to other methods of synchronization, such as timestamp, see "Synchronization techniques" [*MobiLink - Server Administration*].

CHAPTER 7

Tutorial: Using MobiLink with an Adaptive Server Enterprise consolidated database

Contents

Introduction to Adaptive Server Enterprise tutorial

This tutorial shows you how to mobilize an Adaptive Server Enterprise database using MobiLink. It sets up synchronization between an Adaptive Server Enterprise consolidated database and a SQL Anywhere remote database. You could also use UltraLite clients.

The purpose of this tutorial is to mobilize data for a chain of bookstores. Each bookstore in this scenario is a remote synchronization environment. Each bookstore has a local SQL Anywhere database that is synchronized with the Adaptive Server Enterprise database at headquarters. Each bookstore can have several computers that access and manipulate data from the remote database.

Required software

This tutorial assumes you have a complete install of SQL Anywhere (including MobiLink) on your local computer where you are running the tutorial. This tutorial was created using Adaptive Server Enterprise 15.0. The tutorial may work for other versions of Adaptive Server Enterprise, but this is not guaranteed. This tutorial also assumes you have installed Adaptive Server Enterprise 15.0 on your local computer. You may also access Adaptive Server Enterprise 15.0 remotely using Sybase Open Client.

This tutorial assumes that the pubs2 sample schema is installed on an Adaptive Server Enterprise server. The pubs2 sample schema is provided with Adaptive Server Enterprise 15.0 and it is an optional part of the install. For this tutorial, it is used as the consolidated database. Information about this sample can be found in the Adaptive Server Enterprise documentation or can be viewed online at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sqlug/html/sqlug/sqlug894.htm.

This tutorial uses the default **sa** account. When Adaptive Server Enterprise is installed, the **sa** account has a null password. This tutorial assumes you have changed the null password to a valid password. For more information about changing the null password in Adaptive Server Enterprise, see http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sag1/html/sag1/sag1615.htm.

Overview

This tutorial shows you how to:

- Make important considerations, such as synchronization directions for remote tables, when designing a remote schema.
- Add unique primary keys to consolidated and remote databases.
- Create an ODBC data source that connects MobiLink to an Adaptive Server Enterprise database.
- Set up synchronization between a consolidated database and remote database using the **Create Synchronization Model Wizard**.
- Customize a synchronization model using Model mode.
- Deploy a consolidated database and remote database using the **Deploy Synchronization Model Wizard**.
- Synchronize the remote client with the consolidated database.

Suggested background reading

For an overview of MobiLink synchronization, see “Introducing MobiLink synchronization” on page 3.

Lesson 1: Design your schemas

This tutorial assumes that the pubs2 sample schema is installed on an Adaptive Server Enterprise server. The Adaptive Server Enterprise server may be installed locally on your computer or accessed remotely using Sybase Open Client.

The pubs2 sample schema is used as the consolidated database schema. It contains information about stores, titles, authors, publishers, and sales. The following table provides a description of each table in the Adaptive Server Enterprise database:

Table	Description
au_pix	Pictures of the authors.
authors	The authors of the various TITLES in the system.
discounts	Records of various discounts at particular STORES.
sales	Each sale record is one sale made by a particular store.
salesdetail	Contains information about the different TITLES that were included in a particular sale.
stores	Each store record is one store or branch office in the system.
titleauthor	Contains information about which TITLES were written by which AUTHORS.
titles	Records of all of the different books in the system.
blurbs, publishers, and roysched	Contains information that is not needed in this demonstration.

Designing the remote schema

It is unnecessary and inefficient for each store to have a copy of the entire consolidated database. The remote schema uses the same table names, but only contains information relevant to one particular store. To achieve this, the remote schema is designed as a subset of the consolidated database in the following way:

Consolidated table	Remote table
au_pix	Includes all rows.
authors	Includes all rows.
discounts	Filter by stor_id.
sales	Filter by stor_id.
salesdetail	Filter by stor_id.

Consolidated table	Remote table
stores	Filter by stor_id.
titleauthor	Includes all rows.
titles	Includes all rows.
blurbs	Not included on remote.
publishers	Not included on remote.
roysched	Not included on remote.

Each store needs to keep records of all titles and authors so customers can search their inventory. However, a bookstore does not need information about publishers or royalties, so this information is not synchronized to each store. Each store needs information about sales and discounts, but not about sales and discounts related to other stores. This is achieved by filtering rows based on a store identifier.

Note
You can also take a subset of columns from a table if certain columns are not required on the remotes.

The next step is to choose the synchronization direction of each table. You should consider what information a remote database needs to read and what information a remote database needs to create, change, or remove. In this example, a bookstore needs access to the list of authors and titles, but never enters a new author into the system. This places a restriction that authors and titles must always enter the system from the consolidated database at headquarters. However, a bookstore needs to be able to record new sales on a regular basis. These factors lead to the following synchronization directions for the tables:

Table	Synchronization
titleauthor	Download to remote only.
authors	Download to remote only.
au_pix	Download to remote only.
titles	Download to remote only.
stores	Download to remote only.
discounts	Download to remote only.
sales	Download and upload.
salesdetail	Download and upload.

Lesson 2: Prepare the consolidated database

When connecting to the pubs2 database, this tutorial uses the default **sa** account. When Adaptive Server Enterprise is installed, the **sa** account has a null password. This tutorial assumes you have changed the null password to a valid password. For more information about changing the null password in Adaptive Server Enterprise, see http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sag1/html/sag1/sag1615.htm.

In this lesson, you increase the size of the consolidated database for MobiLink synchronization and create unique primary keys.

Increasing the size of the consolidated database

MobiLink needs to add system tables and other objects to the pubs2 database for synchronization. To do this, the size of the pubs2 database must be increased.

To increase the size of the consolidated database

1. Connect to the pubs2 database as **sa**, using the isql utility in Adaptive Server Enterprise. At a command prompt, run the following command, all on one line:

   ```
   isql
   -U sa
   -P Your password for sa account
   -D pubs2
   ```

   If you are accessing Adaptive Server Enterprise remotely, use the -S parameter to specify the server name.

2. To have proper permission for increasing the size of a database, you must access the master database. Run the following command in isql:

   ```
   use master
   ```

3. In Adaptive Server Enterprise, a database is stored on a disk or a portion of a disk. To increase the pubs2 database, run the following command (you must specify the disk where pubs2 is stored):

   ```
   ALTER DATABASE pubs2 ON disk name = 33
   ```

Adding unique primary keys

In a synchronization system, the primary key of a table is the only way to uniquely identify a row in different databases and the only way to detect conflicts. Every table that is being mobilized must have a primary key. The primary key must never be updated. You must also guarantee that a primary key value inserted at one database is not inserted in another database.

There are several ways to generate unique primary keys. For simplicity, the method of composite primary keys is used in this tutorial. This method creates primary keys with multiple columns that are unique across consolidated and remote databases.

To add unique primary keys to the consolidated database

1. Connect to the pubs2 database as **sa**, using the isql utility in Adaptive Server Enterprise. At a command prompt, run the following command, all on one line:

   ```
   isql
   -U sa
   ```

```
-P Your password for sa account
-D pubs2
```

If you are accessing Adaptive Server Enterprise remotely, use the -S parameter to specify the server name.

2. The following rows are not unique based on the composite primary key created for the salesdetail table in step 5. For simplicity, drop the rows by running the following commands:

```
DELETE FROM salesdetail
WHERE stor_id = '5023'
AND ord_num = 'NF-123-ADS-642-9G3'
AND title_id = 'PC8888'

DELETE FROM salesdetail
WHERE stor_id = '5023'
AND ord_num = 'ZS-645-CAT-415-1B2'
AND title_id = 'BU2075'
```

3. The following indexes interfere with the creation of primary keys in step 5. To drop the indexes, run the following commands:

```
DROP INDEX authors.auidind
DROP INDEX titleauthor.taind
DROP INDEX titles.titleidind
DROP INDEX sales.salesind
```

4. Add unique primary keys.

```
ALTER TABLE au_pix ADD PRIMARY KEY (au_id)
ALTER TABLE authors ADD PRIMARY KEY (au_id)
ALTER TABLE titleauthor ADD PRIMARY KEY (au_id, title_id)
ALTER TABLE titles ADD PRIMARY KEY (title_id)
ALTER TABLE discounts ADD PRIMARY KEY (discounttype)
ALTER TABLE stores ADD PRIMARY KEY (stor_id)
ALTER TABLE sales ADD PRIMARY KEY (stor_id, ord_num)
ALTER TABLE salesdetail ADD PRIMARY KEY (stor_id, ord_num, title_id)
```

After running these commands, the MobiLink server should have no trouble connecting to the consolidated database and setting it up for synchronization with any number of remotes.

Note
It is possible to synchronize data with consolidated databases that do not have primary keys. However, you must write you own synchronization events that act on shadow tables that are designed to identify rows uniquely in other tables.

Unique primary keys across all databases

In Lesson 4, the remote schema is created from the consolidated schema. This means that the remote schema has the same primary keys as the consolidated schema.

Columns were specifically chosen to ensure unique primary keys for all databases. For the sales table, the primary key consists of the stor_id and ord_num columns. Any inserted value into the remote sales table must have an unique order number (the stor_id value is always the same). This ensures uniqueness in each remote sales table. The primary key in the consolidated sales table prevents conflicts if multiple stores upload data. Each upload from one store is unique to another store because their stor_id values are different.

For the salesdetail table, the primary key consists of the stor_id, ord_num and title_id columns. There may be multiple book titles in an order. For the remote sales tables, rows may have the same values for stor_id and ord_num, but they must have different title_id values. This ensures uniqueness in each remote salesdetail table. Similar to sales table, each upload to the consolidated database from a store is unique to another store because their stor_id values are different.

Further reading

For more information about Adaptive Server Enterprise issues, see "Adaptive Server Enterprise consolidated database" [*MobiLink - Server Administration*].

For more information about different ways of generating unique primary keys, see "Maintaining unique primary keys" [*MobiLink - Server Administration*].

Lesson 3: Connect with MobiLink

In this lesson, you create an ODBC data source that connects MobiLink to the consolidated database.

To connect MobiLink to the consolidated database

1. Create an ODBC data source.

   You should use the ODBC driver provided by Adaptive Server Enterprise. For this tutorial, use the following configuration settings:

General tab fields	Value
Data Source Name	ase_cons
Description	
Server Name (ASE Host Name)	localhost
Server Port	5000
Database Name	pubs2
Logon ID	sa
Use Cursors	not selected

Transaction tab field	Value
Server Initiated Transactions	not selected

2. Test the ODBC connection:
   a. On the **General** tab, click **Test Connection**.

      The Adaptive Server Enterprise logon screen appears.

   b. Enter the password for the **sa** account.

      The **Logon Succeeded** message appears.

After configuring your ODBC data source, you can use the MobiLink plug-in to connect to the consolidated database and create a synchronization model.

Further reading

For more information about recommended ODBC drivers for MobiLink, see Recommended ODBC Drivers for MobiLink.

Lesson 4: Create a synchronization model

The **Create Synchronization Model Wizard** takes you through the steps of setting up synchronization between the consolidated database and remote database.

To create a synchronization model

1. Start Sybase Central.
2. From the **Tools** menu, choose **MobiLink 11** » **Set Up MobiLink Synchronization**.

   The **Create Synchronization Model Wizard** appears.
3. Type **sync_ase** as the name and type the location of your new model.
4. On the **Primary Key Requirements** page, select all three checkboxes. (You guarantee unique primary keys in Lesson 2.)
5. On the **Consolidated Database Schema** page, connect to the consolidated database:
   a. Click **Choose The Consolidated Database**.

      The **Connect To Consolidated Database** window appears.
   b. Select ODBC Data Source Name and choose **ase_cons**.
   c. Type **sa** as the **User Id** and the **Password** for the **sa** account. Click **OK**.
   d. If this is the first time the consolidated database has been used by MobiLink, a window appears. It asks if you want to install the MobiLink system setup. Installing the MobiLink system setup adds MobiLink system tables and procedures. Click **Yes**.

      If you see an error message, ensure that the data source is configured properly.
   e. Once the MobiLink system tables and procedures are added to the consolidated database, the name, user, product and version appear on the page. Click **Next**.
6. Create a remote schema:
   a. On the **Remote Database Schema** page, select **No**, **Create A New Remote Database Schema** and click **Next**.
   b. On the **New Remote Database Schema** page, select the checkboxes for the following tables to include them in the remote schema and then click **Next**.

      - au_pix
      - authors
      - discounts
      - sales
      - salesdetail
      - stores
      - titleauthor
      - titles
7. Choose and configure the download type:
   a. On the **Download Type** page, select **Timestamp-based Download**.

      Choosing timestamp-based downloads minimizes the amount of data that is transferred because only data that has been updated since the last download is transmitted.

b. On the **Timestamp Download Options** page, select **Use Shadow Tables To Hold Timestamp Columns**.

Using shadow tables is often preferred because it does not require any changes to existing tables.

8. On the **Download Deletes** page, specify how to propagate record deletions to remote devices:
   a. To indicate that you want the remote database to download deletes, select **Yes**.
   b. Select **Use Shadow Tables To Record Deletions**.

   MobiLink creates shadow tables on the consolidated database to implement deletions.
9. On the **Download Subset** page, specify that the remote database should download only a subset of data from the consolidated database:
   a. Select **Yes, Download the Same Data to Each Remote**. (In Step 2 of the Model mode lesson, you specify how to download specific data to a remote database by using custom logic.)
10. On the **Upload Conflict Detection** page, select **No Conflict Detection**.

    Many applications require conflict detection, but this tutorial uses no conflict detection.
11. On the **Publication, Script Version and Description** page, type **sync_ase_publication** as your publication and type **sync_ase_scriptversion** as your script version.

    The publication is the object on the remote database that specifies what data is synchronized. MobiLink server scripts define how uploaded data from remotes should be applied to the consolidated database, and script versions group scripts. You can use different script versions for different applications, allowing you to maintain a single MobiLink server while synchronizing different applications.
12. Click **Finish**.

    Your model appears in Model mode.

Model Mode

1. To specify the direction data is synchronized, set the directions in the **Mapping Direction** column as follows:
   - Sales and salesdetail are bi-directional (both upload and download).
   - The remaining tables are download only.
2. Filter the rows downloaded to the remote database by remote ID:
   a. For the stores table, change the **Download Subset** to **Custom**.
   b. Open the **Download Subset** tab at the bottom of the screen.
   c. The remote ID uniquely identifies a remote database. To filter rows by remote ID, add a restriction to the WHERE clause of the download_cursor script. To do this, type the following search condition in the **SQL Expression To Use In The Download Cursor's WHERE Clause** text box:

      ```
      "dbo"."stores"."stor_id" = {ml s.remote_id}
      ```

      The download cursor script specifies what columns and rows are downloaded from each table to the remote database. The search condition ensures that you only download information about one store, namely, the store that has an identifier that equals the remote ID for the database.

d. Follow these steps for the sales, salesdetail, and discounts tables. Make sure you rename the table in the expression appropriately.

3. Save the synchronization model.

   The synchronization model is complete and ready to be deployed.

Further reading

- "Setting up a consolidated database" [*MobiLink - Server Administration*]
- "MobiLink server system tables" [*MobiLink - Server Administration*]
- "MobiLink system procedures" [*MobiLink - Server Administration*]
- "Writing download_delete_cursor scripts" [*MobiLink - Server Administration*]
- "Handling conflicts" [*MobiLink - Server Administration*]
- "Resolving conflicts" [*MobiLink - Server Administration*]
- "Publishing data" [*MobiLink - Client Administration*]
- "Model mode" on page 32
- "Modifying the download type" on page 34
- "Modifying conflict detection and resolution" on page 38
- "Modifying table and column mappings" on page 32

Lesson 5: Deploy the synchronization model

The **Deploy Synchronization Model Wizard** allows you to deploy the consolidated database and remote database. You can deploy each of these individually or you can deploy both of them. The **Deploy Synchronization Model Wizard** takes you through the steps of configuring options for deployment.

To deploy the synchronization model

1. In the left pane of Sybase Central, right-click the synchronization model you just created and choose **Deploy** from the popup menu.

   The **Deploy Synchronization Model Wizard** appears.

2. Choose deployment settings:
   a. Select **Specify The Deployment Details For One Or More Of The Following**.
   b. Select **Consolidated Database**, **Remote Database And Synchronization Client**, and **MobiLink Server**.

      This instructs MobiLink to deploy each piece of the synchronization environment.

3. On the **Consolidated Database Deployment Destination** page, configure the options for deploying the consolidated database:
   a. Select **Save Changes To The Following SQL File**. For this tutorial, use the default location.

      MobiLink generates a *.sql* file that makes changes to the consolidated database to set up for synchronization.
   b. You may select **Connect To The Consolidated Database To Directly Apply The Changes**. This option causes MobiLink to apply the changes immediately to the consolidated database.

      You may also examine the *.sql* file later and make your own changes. Then, you must run the *.sql* file yourself.

4. On the same page, connect to the consolidated database:
   a. Click **Choose A Consolidated Database**.

      The **Connect To Consolidated Database** window appears.
   b. Select **ODBC Data Source Name** and choose **ase_cons**.
   c. Type **sa** as the **User Id** and the **Password** for the **sa** account. Click **OK**.

      The **Checking For MobiLink System Setup** window appears. Once the tables from the consolidated database have loaded, the name, user, product and version appear on the page.
   d. Click **Next**.

      A window appears and it asks if you want create the *consolidated* directory. Click **Yes**.

5. On the **Remote Database Deployment** page, specify the type of remote database being deployed. Select **New SQL Anywhere Database**.

6. On the **New SQL Anywhere Remote Database** page, configure the options for creating a new remote database:
   a. Select **Make A Command File And A SQL File With Commands To Create The Database**.

MobiLink generates another *.sql* file with the commands to set up the remote database with all schema and synchronization information.

b. Use the default location of the SQL file.

c. Select **Create A Remote SQL Anywhere Database**.

If you choose not to do this, you must generate a new remote database, and then run the *.sql* file against it. This allows you to examine the *.sql* file later and make your own changes.

d. Use the default location of the remote SQL Anywhere database.

e. Click **Next**.

A window appears and it asks if you want to create the *remote* directory. Click **Yes**.

7. Configure the options for deploying the MobiLink server:

a. On the **MobiLink User** page, type **ase_remote** as the MobiLink user name and type **ase_pass** as the password.

The MobiLink user is stored in a MobiLink system table and is used to authenticate the user during synchronization.

b. On the **Synchronization Stream Parameters** page, choose TCP/IP as the communication protocol and enter 2439 as the port to use.

c. On the **Client Stream Parameters** page, type **localhost** as the host name of the MobiLink server computer.

You can optionally type your computer's name or IP address, the name or IP address of another network server you want to use, or other client stream options.

d. On the **MobiLink Server Stream Parameters** page, you can specify additional server stream options.

e. On the **Verbosity For MobiLink Server** page, choose the amount of log verbosity and use the default file name for the log file.

You can use the log file for troubleshooting if you encounter any errors.

f. On the **Options For MobiLink Server** page, type **ase_mlsrv** as the name of the MobiLink server and use the default location of the command file.

You use the command file to start the MobiLink server.

g. Click **Next**.

A window appears and it asks if you want to create the *mlsrv* directory. Click **Yes**.

8. Configure the options for deploying the remote synchronization client:

a. On the **Verbosity For SQL Anywhere Remote Synchronization Client** page, choose the verbosity.

b. On the same page, use the default file name of the remote database log file.

c. On the **Advanced Options For SQL Anywhere Remote Synchronization Client** page, you can specify advanced options.

d. On the same page, use the default location of the command file and click **Next**.

MobiLink generates a command file that you can use to start synchronization.

9. Click **Finish**.

A **Completing The Deploy Wizard** window appears. Click **Finish**.

10. A **Deploying** window appears. When the deployment completes, it shows Status: Completed. Click **Close**.

Your consolidated database is fully configured for synchronization with many remote clients, and you have successfully deployed one remote client. If you want to deploy other remote clients, you can run this wizard again, making sure to create a new MobiLink user and opt out of deploying the consolidated database and MobiLink server. Since these have already been deployed, all you need to do is deploy other remote synchronization clients.

Further reading

- "Deploying models" on page 43
- "Creating a remote database" [*MobiLink - Client Administration*]
- "Introduction to MobiLink users" [*MobiLink - Client Administration*]

Lesson 6: Start the server and client

In this lesson, you start the MobiLink server and remote database.

Previously, you modified the download cursor script to download information related to one store. In this lesson, you specify the store by setting the remote ID to the store identifier.

To start the MobiLink server

1. At a command prompt, navigate to the folder where you created the synchronization model. (This is the root directory you chose in the first step of the **Synchronization Model Wizard**.)

   If you used the suggested directory names, you should navigate to the following directory: *sync_ase\mlsrv*.

2. To start the MobiLink server, run the following command:

   ```
   sync_ase_mlsrv.bat "dsn=ase_cons;uid=sa;pwd=your password for sa account;"
   ```

   - **sync_ase_mlsrv.bat** is the command file to start the MobiLink server.
   - **dsn** is your ODBC data source name.
   - **uid** is the user name you use to connect to the consolidated database (the default for Adaptive Server Enterprise is **sa**).
   - **pwd** is the password you use to connect as **sa**.

   A MobiLink server messages window appears. When this command runs successfully, it shows MobiLink Server Started.

   If the MobiLink server fails to start, check connection information for your consolidated database.

To start the remote database

1. At a command prompt, navigate to the directory where the **Deploy Synchronization Model Wizard** created your remote database.

   If you used the suggested directory names, you navigate to the following directory: *sync_ase\remote*.

2. To start your remote SQL Anywhere database, run the following command:

   ```
   dbeng11 -n remote_eng sync_ase_remote.db -n remote_db
   ```

   - **dbeng11** is the database server used to start the SQL Anywhere database.
   - **remote_eng** is the database server name.
   - **sync_ase_remote.db** is the database file that is started on remote_eng.
   - **remote_db** is the name of the database on remote_eng.

   A SQL Anywhere messages window appears. When this command runs successfully, a SQL Anywhere database server named remote_eng starts and loads the database called remote_db.

Set the remote ID

In the remote schema, each remote database represents one store. The synchronization scripts you wrote include logic that instructs the MobiLink server to download a subset of data based on the remote ID of the remote database. You must set the database's remote ID to the value of a valid store identifier.

It is important to complete this step before the first synchronization because when the remote device synchronizes for the first time, it downloads all information related to the store (in this case, Thoreau Reading Discount Chain).

To set the remote ID to a valid store identifier

1. Choose a valid store identifier.
   a. Connect to the pubs2 database as **sa**, using the isql utility in Adaptive Server Enterprise. At a command prompt, run the following command, all on one line:

      ```
      isql
      -U sa
      -P Your password for sa account
      -D pubs2
      ```

      If you are accessing Adaptive Server Enterprise remotely, use the -S parameter to specify the server name.

   b. To view a list of valid store identifiers in the stores table, execute the following statement:

      ```
      SELECT * FROM stores
      ```

      In this tutorial, the remote database represents the Thoreau Reading Discount Chain store, which has a value of 5023 for its store identifier.

   c. To exit Adaptive Server Enterprise, run the following command:

      ```
      exit
      ```

2. To set the database's remote ID to 5023, run the following command, all on one line:

   ```
   dbisql
      -c "eng=remote_eng;dbn=remote_db;uid=DBA;pwd=sql"
      "SET OPTION PUBLIC.ml_remote_id='5023'"
   ```

   - **dbisql** is the application used to execute SQL commands against a SQL Anywhere database.
   - **eng** specifies the database server name remote_eng.
   - **dbn** specifies the database name remote_db.
   - **uid** is the user name used to connect to your remote database.
   - **pwd** is the password used to connect to your remote database.
   - **SET OPTION PUBLIC.ml_remote_id='5023'** is the SQL command used to set the remote ID to 5023.

Further reading

- "The SQL Anywhere database server" [*SQL Anywhere Server - Database Administration*]
- "Synchronizing a deployed model" on page 45
- "Running the MobiLink server" [*MobiLink - Server Administration*]
- "Remote IDs" [*MobiLink - Client Administration*]

Lesson 7: Synchronize

Now you are ready to synchronize the remote client for the first time. This is done with the MobiLink client program dbmlsync. Dbmlsync connects to the remote database, authenticates itself with the MobiLink server, and performs all of the uploads and downloads necessary to synchronize the remote and consolidated databases based on a publication in the remote database.

To synchronize the remote client

- At a command prompt, run the following command, all on one line:

```
dbmlsync
   -c "eng=remote_eng;dbn=remote_db;uid=DBA;pwd=sql;"
   -n sync_ase_publication
   -u ase_remote -mp ase_pass
```

  - **dbmlsync** is the synchronization application.
  - **eng=remote_eng** specifies the name of the remote database server.
  - **dbn=remote_db** specifies the name of the remote database.
  - **uid** is the user name used to connect to the remote database.
  - **pwd** is the password used to connect to the remote database.
  - **sync_ase_publication** is the name of the publication on the remote device that is used to perform the synchronization. (This publication was created by the **Create Synchronization Model Wizard**.)
  - **ase_remote** is the user name used to authenticate with the MobiLink server.
  - **ase_pass** is the password used to authenticate with the MobiLink server.

  A SQL Anywhere MobiLink Client messages window appears. The messages window shows you the progress. When this command runs successfully, the dbmlsync application populates the remote database with a subset of information from the consolidated database.

  If synchronization fails, you can start by checking the connection information you pass to the dbmlsync application, as well as the MobiLink user name and password. Failing that, check the publication name you used, and ensure the consolidated database and MobiLink server are running. You can also examine the contents of the synchronization logs (server and client).

> **Note**
> If you are running the dbmlsync application on a different computer from your MobiLink server, you must also pass in arguments that specify the location of the MobiLink server.

View the data

After successfully synchronizing the remote client to the consolidated database through the MobiLink server, the remote data should be populated with information relevant to one store. You can verify this in Sybase Central using the SQL Anywhere 11 plug-in.

To view the data in the remote database

1. Start Sybase Central.

2. Connect to the remote database:
   a. On the left pane, right-click **SQL Anywhere 11** and choose **Connect**.

      A **Connect** window appears.
   b. Type **DBA** as the **User Id** and **sql** as the **Password**.
   c. Under the **Identification** tab, type **remote_eng** as the **Server Name** and **remote_db** as the **Database Name**.
   d. Click **OK**.
3. If the tables created from the consolidated database are not visible, perform the following steps:
   a. Right-click *remote_db* and **Configure Owner Filter**.

      The **Configure Owner Filter** window appears.
   b. Select **dbo** and click **OK**.

      The tables created from the consolidated database appear in the left pane. Ownership of these tables by dbo is preserved in the remote database.
4. Choose any remote table and then click the **Data** tab on the right pane.

   In the sales, salesdetail and stores tables, all the records are for the store with an identifier of 5023. This particular store is not concerned with the sales information of other stores. For this reason, you set the synchronization scripts to filter out rows by the remote ID, and you set this database's remote ID to the value of a particular store identifier. Now this particular store's database takes up less space, and requires less time to synchronize. Since the remote database size is kept to a minimum, frequently performed operations such as entering a new sale or processing a refund on a previous sale run faster and more efficiently.

Further reading

- "The synchronization process" on page 16
- "dbmlsync syntax" [*MobiLink - Client Administration*]

Cleanup

Regenerate the pubs2 database and remove all tutorial materials from your computer.

To regenerate the pubs2 database

- To run the script that installs the pubs2 database, run the following command:

```
isql
-U sa
-P Your password for sa account
-i %SYBASE%\%SYBASE_ASE%\scripts\instpbs2
```

  If you are accessing Adaptive Server Enterprise remotely, use the -S parameter to specify the server name. You also have to copy the instpbs2 file locally onto your computer. The -i parameter needs to updated so that the new location of the instpbs2 file is specified.

To delete the synchronization model

1. Start Sybase Central.
2. Double-click **MobiLink 11** in the right pane.

   The **sync_ase** model appears.
3. Right click **sync_ase** and choose **Delete**.

   The **Confirm Delete** window appears. Choose **Delete**.

To erase the remote database

- To erase the remote database, use the dberase utility. Run the following command:

```
dberase sync_ase\remote\sync_ase_remote.db
```


CHAPTER 8

Tutorial: Using Java synchronization logic

Contents

Introduction to Java Synchronization tutorial

This tutorial guides you through the basic steps for using Java synchronization logic. Using the CustDB sample as a SQL Anywhere consolidated database, you specify simple class methods for MobiLink table-level events. The process also involves running the MobiLink server (mlsrv11) with an option to set the path of compiled Java classes.

Required software

- SQL Anywhere 11
- Java Software Development Kit

Competencies and experience

You require:

- familiarity with Java
- basic knowledge of MobiLink event scripts

Goals

You gain competence and familiarity with:

- using simple Java class methods for MobiLink table-level events

Key concepts

This section uses the following steps to implement basic Java-based synchronization using the MobiLink CustDB sample database:

- compiling a source file with MobiLink server API references
- specifying class methods for particular table-level events
- running the MobiLink server (mlsrv11) with the -sl java option
- testing synchronization with a sample Windows client application

Suggested background reading

For more information about synchronization scripts, see "Introduction to synchronization scripts" [*MobiLink - Server Administration*].

Lesson 1: Compile the CustdbScripts Java class

Java classes encapsulate synchronization logic in methods.

In this lesson, you compile a class associated with the CustDB sample database.

MobiLink Database Sample

SQL Anywhere ships with a SQL Anywhere sample database (CustDB) that is already set up for synchronization, including the SQL scripts required for synchronization. The CustDB ULCustomer table, for example, is a synchronized table that supports a variety of table-level events.

CustDB is designed to be a consolidated database server for both UltraLite and SQL Anywhere clients. The CustDB database has a DSN called SQL Anywhere 11 CustDB.

CustdbScripts class

In this section, you create a Java class called CustdbScripts with logic to handle the ULCustomer upload_insert and download_cursor events. You enter the CustdbScripts code in a text editor and save the file as *CustdbScripts.java*.

To create CustdbScripts.java

1. Create a directory for the Java class and assembly.

   This tutorial assumes the path *c:\mljava*.

2. Using a text editor, enter the CustdbScripts code:

```
public class CustdbScripts {
    public static String UploadInsert() {
        return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)");
    }
    public String DownloadCursor(java.sql.Timestamp ts,String user ) {
        return(
           "SELECT cust_id, cust_name
            FROM ULCustomer where last_modified >= ' " + ts + " ' ");
    }
}
```

   Note
   When creating your own custom scripts, make sure that the classes and associated methods are defined as **public**.

3. Save the file as *CustdbScripts.java* in *c:\mljava*.

Compiling the Java source code

To execute Java synchronization logic, the MobiLink server must have access to the classes in *mlscript.jar*. This jar file contains a repository of MobiLink server API classes to use in your Java methods.

When compiling Java source code for MobiLink, you must include *mlscript.jar* to make use of the MobiLink server API. In this section, you use the javac utility's -classpath option to specify *mlscript.jar* for the CustdbScripts class.

To compile the Java source code (Windows)

1. At a command prompt, navigate to the folder containing *CustdbScripts.java* (*c:\mljava*).
2. Run the following command.

```
javac custdbscripts.java -classpath "%sqlany11%\java\mlscript.jar"
```

The *CustdbScripts.class* file is generated.

Further reading

For more information about the MobiLink server API for Java, see "MobiLink server API for Java Reference" [*MobiLink - Server Administration*].

For more information about Java methods, see "Methods" [*MobiLink - Server Administration*].

For more information about the CustDB sample database, and using alternate RDBMS servers, see "Setting up the CustDB consolidated database" on page 60.

Lesson 2: Specify class methods to handle events

CustdbScripts.class, created in the previous lesson, encapsulates the methods UploadInsert and DownloadCursor. These methods contain implementations for the ULCustomer upload_insert and download_cursor events, respectively.

In this section, you specify class methods for table-level events using two approaches:

- Using the MobiLink Admin mode in Sybase Central:

  Connect to the CustDB database with Sybase Central, change the language for the upload_insert script to Java, and specify CustdbScripts.UploadInsert to handle the event.

- Using the ml_add_java_table_script stored procedure:

  Connect to the CustDB database with Interactive SQL and execute ml_add_java_table_script, specifying CustdbScripts.DownloadCursor to handle the download_cursor event.

To specify CustdbScripts.UploadInsert to handle the ULCustomer upload_insert event

1. Connect to the sample database using the Sybase Central MobiLink Admin mode:
   - Start Sybase Central.
   - From the **View** menu, ensure that **Folders** is selected.
   - From the **Connections** menu, choose **Connect With MobiLink 11**.
   - On the **Identification** tab, choose the **ODBC Data Source Name** SQL Anywhere 11 CustDB.
   - Click **OK** to Connect.
   - Sybase Central should now display the CustDB data source under the MobiLink 11 plug-in.
2. Delete the existing upload_insert event for the ULCustomer table:
   - In the left pane, open the **Synchronized Tables** folder and select the ULCustomer table. A list of table-level scripts appears in the right pane.
   - Select the table script associated with the custdb 11.0 upload_insert event. From the **Edit** menu, choose **Delete**. The **Confirm Delete** window appears. You must confirm that you want to delete the object.
   - Select **Yes**. The custdb 11.0 upload_insert event is removed from the ULCustomer table.
3. Create a new upload_insert event for the ULCustomer table:
   - With the ULCustomer table selected in the **Synchronized Tables** folder, choose **File** » **New** » **Table Script**.
   - Select custdb 11.0 as the script version.
   - Select upload_insert as the event to create and click **Next**.
   - Select **Create A New Script Definition**, and choose **Java**.
   - Click **Finish**.
4. Instruct the MobiLink server to run the **CustdbScripts.UploadInsert** method on an upload_insert event.
   - Select the custDB 11.0 - upload_insert script.
   - In the right pane, enter the following code:

```
CustdbScripts.UploadInsert
```

- From the **File** menu, choose **Save**.

5. Exit Sybase Central.

This step used Sybase Central to specify a Java method as the script for the ULCustomer upload_insert event.

Alternatively, you can use the ml_add_java_connection_script and ml_add_java_table_script stored procedures. Using these stored procedures is more efficient if you have a large number of Java methods to handle synchronization events.

See "ml_add_java_connection_script system procedure" [*MobiLink - Server Administration*] and "ml_add_java_table_script system procedure" [*MobiLink - Server Administration*].

To specify CustdbScripts.DownloadCursor() to handle the ULCustomer download_cursor event

1. Connect to the sample database with Interactive SQL.
   - Open Interactive SQL.

     Choose **Start** » **Programs** » **SQL Anywhere 11** » **Interactive SQL**, or run the following command:

     ```
     dbisql
     ```

     The Connect window appears.
   - On the **Identification** tab, choose the **ODBC Data Source Name** SQL Anywhere 11 CustDB.
   - On the **Database** tab, ensure that the following option is not selected: **Search Network For Database Servers**.
   - Click **OK** to connect.
2. Run the following command in Interactive SQL:

   ```
   CALL ml_add_java_table_script(
   'custdb 11.0',
   'ULCustomer',
   'download_cursor',
   'CustdbScripts.DownloadCursor');
   COMMIT;
   ```

   Following is a description of each parameter:

Parameter	Description
custdb 11.0	The script version.
ULCustomer	The synchronized table.
download_cursor	The event name.
CustdbScripts.DownloadCursor	The fully qualified Java method.

3. Exit Interactive SQL.

In this lesson, you specified your Java methods to handle ULCustomer table-level events. The next lesson ensures that the MobiLink server loads the appropriate class files and the MobiLink server API.

Further reading

For more information about adding scripts with stored procedures, see "ml_add_java_connection_script system procedure" [*MobiLink - Server Administration*] and "ml_add_java_table_script system procedure" [*MobiLink - Server Administration*].

For general information about adding and deleting synchronization scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

Lesson 3: Run the MobiLink server with -sl java

Running the MobiLink server with the -sl java -cp option specifies a set of directories to search for class files, and forces the Java Virtual Machine to load on server startup.

To start the MobiLink server (mlsrv11) and load Java assemblies

- At a command prompt, run the following command:

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -sl java (-cp c:\mljava)
  ```

  A message appears that the server is started. Now the Java method is executed when the ULCustomer table upload_insert event triggers during synchronization.

Further reading

For more information, see "-sl java option" [*MobiLink - Server Administration*].

Lesson 4: Test synchronization

UltraLite comes with a sample Windows client that automatically invokes the dbmlsync utility when the user issues a synchronization. It is a simple sales-status application that you can run against the CustDB consolidated database you started in the previous lesson.

Start the application (Windows)

To start and synchronize the sample application

1. Launch the sample application.

   From the **Start** menu, choose **Programs** » **SQL Anywhere 11** » **UltraLite** » **Windows Sample Application**.

2. Enter an employee ID.

   Type a value of **50** and press Enter.

The application automatically synchronizes and a set of customers, products, and orders are downloaded to the application from the CustDB consolidated database.

Add an order (Windows)

To add an order

1. From the **Order** menu, choose **New**.

   The **Add New Order** screen appears.

2. Enter a new Customer Name.

   For example, enter **Frank Javac**.

3. Choose a product, and enter the quantity and discount.
4. Press Enter to add the new order.

You have now modified the data in your local UltraLite database. This data is not shared with the consolidated database until you synchronize.

To synchronize with the consolidated database and trigger the upload_insert event

- From the **File** menu, choose **Synchronize**.

A window appears indicating one Insert successfully uploaded to the consolidated database.

Further reading

For more information about the CustDB Windows Application, see "Exploring the CustDB sample for MobiLink" on page 57.

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Return the ULCustomer table upload_insert and download_cursor scripts to their original SQL logic.
   - Open Interactive SQL.

     Choose **Start** » **Programs** » **SQL Anywhere 11** » **Interactive SQL**, or run the following command:

     ```
     dbisql
     ```

     The **Connect** window appears.
   - On the **Identification** tab, choose the **ODBC Data Source Name** SQL Anywhere 11 CustDB.
   - Click **OK** to Connect.
   - Run the following commands in Interactive SQL:

     ```
     CALL ml_add_table_script( 'custdb 11.0',
        'ULCustomer',
        'upload_insert',
        'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

     CALL ml_add_table_script( 'custdb 11.0',
        'ULCustomer',
        'download_cursor',
        'SELECT "cust_id", "cust_name"
         FROM "ULCustomer" WHERE "last_modified" >= ?' );
     COMMIT;
     ```
2. Close the SQL Anywhere, MobiLink, and synchronization client windows by right-clicking each taskbar item and choosing **Close**.
3. Delete all tutorial-related Java sources.

   Delete the folder containing your *CustdbScripts.java* and *CustdbScripts.class* files (*c:\mljava*).

> **Note**
> Ensure that you do not have any important files in *c:\mljava*.

Further reading

For more information about writing MobiLink synchronization scripts in Java, see "Setting up Java synchronization logic" [*MobiLink - Server Administration*].

For an example illustrating the use of Java synchronization scripts for custom authentication, see "Java synchronization example" [*MobiLink - Server Administration*].

For more information about synchronization scripting, see "Writing synchronization scripts" [*MobiLink - Server Administration*] and "Synchronization events" [*MobiLink - Server Administration*].

For an introduction to other methods of synchronization such as timestamp, see "Synchronization techniques" [*MobiLink - Server Administration*].

CHAPTER 9

Tutorial: Using .NET synchronization logic

Contents

Introduction to .NET synchronization tutorial

This tutorial guides you through the basic steps for using .NET synchronization logic. Using the CustDB sample as a SQL Anywhere consolidated database, you specify simple class methods for MobiLink table-level events. The process also involves running the MobiLink server (mlsrv11) with an option that sets the path of .NET assemblies.

Required software

- SQL Anywhere 11
- Microsoft .NET Framework SDK

Competencies and experience

You should have:

- familiarity with .NET
- basic knowledge of MobiLink event scripts

Goals

You gain competence and familiarity with:

- utilizing .NET class methods for MobiLink table-level event scripts

Key concepts

This section uses the following steps to implement basic .NET synchronization using the MobiLink CustDB sample database:

- compiling the *CustdbScripts.dll* private assembly with MobiLink references
- specifying class methods for table-level events
- running the MobiLink server (mlsrv11) with the -sl dnet option
- testing synchronization with a sample Windows client application

Suggested background reading

For more information about synchronization scripts, see "Introduction to synchronization scripts" [*MobiLink - Server Administration*].

Lesson 1: Compile the CustdbScripts.dll assembly with MobiLink references

.NET classes encapsulate synchronization logic in methods.

In this lesson, you compile a class associated with the CustDB sample database.

MobiLink database sample

SQL Anywhere ships with a SQL Anywhere sample database (CustDB) that is already set up for synchronization, including the SQL scripts required to drive synchronization. The CustDB ULCustomer table, for example, is a synchronized table supporting a variety of table-level events.

CustDB is designed to be a consolidated database server for both UltraLite and SQL Anywhere clients. The CustDB database has a DSN called SQL Anywhere 11 CustDB.

The CustdbScripts Assembly

In this section, you create a .NET class called CustdbScripts with logic to handle the ULCustomer upload_insert and download_cursor events.

MobiLink server API

To execute .NET synchronization logic, the MobiLink server must have access to the classes in *iAnywhere.MobiLink.Script.dll*. *iAnywhere.MobiLink.Script.dll* contains a repository of MobiLink server API for .NET classes to use in your .NET methods.

For more information about the MobiLink server API for .NET, see "MobiLink server API for .NET reference" [*MobiLink - Server Administration*].

When compiling the CustdbScripts class, you must include this assembly to make use of the API. You can compile your class using Visual Studio .NET or at a command prompt.

- In Visual Studio .NET, create a new Class Library and enter the CustdbScripts code. Link *iAnywhere.MobiLink.Script.dll*, and build the assembly for your class.
- Put the CustdbScripts code in a text file and save the file as *CustdbScripts.cs* (or *CustdbScripts.vb* for Visual Basic .NET). Using a command line compiler, reference *iAnywhere.MobiLink.Script.dll* and build the assembly for your class.

To create the CustdbScripts assembly using Visual Studio .NET

1. Start a new Visual C# or Visual Basic .NET Class Library project.

   Use CustdbScripts for the project Name and enter an appropriate path. This tutorial assumes the path *c:\mldnet*.

2. Enter the CustdbScripts code.

   For C#, type:

```
namespace MLExample
{
class CustdbScripts
{
  public static string UploadInsert()
```

```
 {
   return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)");
 }
 public static string DownloadCursor(System.DateTime ts, string user )
 {
   return("SELECT cust_id, cust_name
           FROM ULCustomer
            WHERE last_modified >= '" + ts.ToString("yyyy-MM-dd
hh:mm:ss.fff") +"'");
 }
}
}
```

For Visual Basic .NET, type:

```
Namespace MLExample

 Class CustdbScripts

  Public Shared Function UploadInsert() As String
    Return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)")
  End Function

  Public Shared Function DownloadCursor(ByVal ts As System.DateTime, ByVal
user As String) As String
   Return("SELECT cust_id, cust_name FROM ULCustomer " + _
    "WHERE last_modified >= '" + ts.ToString("yyyy-MM-dd hh:mm:ss.fff")
+"'")
  End Function

 End Class

End Namespace
```

3. Add a reference to the MobiLink server API.
   - From the Visual Studio **Project** menu, choose **Add Existing Item**.
   - Select *iAnywhere.MobiLink.Script.dll* in the *Assembly\v2* subdirectory of your SQL Anywhere installation. In Visual Studio .NET, from the **Open** menu, choose **Link File**. In Visual Studio 2005, from the **Add** menu choose **Add Link**.
4. Right-click the CustdbScripts project and select the table **Common Properties** » **General**. Make sure that the text field **Root Namespace** is cleared of all text.
5. Build *CustdbScripts.dll*.

   From the **Build** menu, choose **Build CustdbScripts**.

   This creates *CustdbScripts.dll* in *C:\mldnet\CustdbScripts\CustdbScripts\bin\Debug*.

To create the CustdbScripts assembly at a command prompt

1. Create a directory for the .NET class and assembly.

   This tutorial assumes the path *c:\mldnet*.
2. Using a text editor, enter the CustdbScripts code.

   For C#, type:

```
namespace MLExample
{
```



```
class CustdbScripts
{
  public static string UploadInsert()
 {
   return("INSERT INTO ulcustomer(cust_id,cust_name) values (?,?)");
 }
 public static string DownloadCursor(System.DateTime ts, string user )
 {
   return("SELECT cust_id, cust_name FROM ULCustomer where last_modified
>= '" + ts.ToString("yyyy-MM-dd hh:mm:ss.fff") +"'");
 }
}
}
```

For Visual Basic .NET, type:

```
Namespace MLExample

 Class CustdbScripts

  Public Shared Function UploadInsert() As String
    Return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)")
  End Function

  Public Shared Function DownloadCursor(ByVal ts As System.DateTime, ByVal
user As String) As String
   Return("SELECT cust_id, cust_name FROM ULCustomer " + _
    "WHERE last_modified >= '" + ts.ToString("yyyy-MM-dd hh:mm:ss.fff")
+"'")
  End Function

 End Class

End Namespace
```

3. Save the file as *CustdbScripts.cs* (*CustdbScripts.vb* for Visual Basic .NET) in *c:\mldnet*.
4. Compile the file using the following command.

   For C#, type:

   ```
   csc /out:c:\mldnet\custdbscripts.dll /target:library /
   reference:"%sqlany11%\Assembly\v2\iAnywhere.MobiLink.Script.dll" c:\mldnet
   \CustdbScripts.cs
   ```

   For Visual Basic .NET, type:

   ```
   vbc /out:c:\mldnet\custdbscripts.dll /target:library /
   reference:"%sqlany11%\Assembly\v2\iAnywhere.MobiLink.Script.dll" c:\mldnet
   \CustdbScripts.vb
   ```

   The *CustdbScripts.dll* assembly is generated.

Further reading

For more information about the MobiLink server API for .NET, see “MobiLink server API for .NET reference” [*MobiLink - Server Administration*].

For more information about .NET methods, see “Methods” [*MobiLink - Server Administration*].

Lesson 2: Specify class methods for events

For more information about the CustDB sample database, see "Setting up the CustDB consolidated database" on page 60.

CustdbScripts.dll, created in the previous lesson, encapsulates the methods UploadInsert() and DownloadCursor(). These methods contain implementation for the ULCustomer upload_insert and download_cursor events, respectively.

In this section, you specify class methods for table-level events using two approaches:

1. Using the MobiLink Synchronization plug-in.

   You connect to the CustDB database with Sybase Central, change the language for the upload_insert script to .NET, and specify MLExample.CustdbScripts.UploadInsert to handle the event.

2. Using the ml_add_dnet_table_script stored procedure.

   You connect to the CustDB database with Interactive SQL and execute ml_add_dnet_table_script, specifying MLExample.CustdbScripts.DownloadCursor for the download_cursor event.

To subscribe CustdbScripts.uploadInsert() to the upload_insert event for the ULCustomer table

1. Connect to the sample database using the MobiLink Synchronization plug-in:
   - Start Sybase Central.
   - In the **View** menu, ensure that **Folders** is selected.
   - From the **Connections** menu, select **Connect With MobiLink 11**.
   - On the **Identification** tab, choose the **ODBC Data Source Name** SQL Anywhere 11 CustDB.
   - Click **OK** to Connect.
   - Sybase Central should now display the CustDB data source under the MobiLink 11 plug-in.
2. Change the language for the ULCustomer table upload_insert event to .NET:
   - In the left pane, open the **Synchronized Tables** folder and select the ULCustomer table. A list of table-level scripts appears in the right pane.
   - In the right pane, open the custdb 11.0 upload_insert table script. From the **File** menu, choose **Language**, and change the language to .NET.
3. Enter the .NET method name for the custdb 11.0 upload_insert table script.
   - Delete the contents of the upload_insert table script so that the right pane appears blank.
   - Type the following in the right pane:

     ```
     MLExample.CustdbScripts.UploadInsert
     ```

   - From the **File** menu, choose **Save** to save the script.
4. Exit Sybase Central.

This step uses Sybase Central to specify a .NET method as the script for the ULCustomer upload_insert event.

Alternatively, you can use the ml_add_dnet_connection_script and ml_add_dnet_table_script stored procedures. Using these stored procedures is more efficient, especially if you have a large number of .NET methods to handle synchronization events.

See "ml_add_dnet_connection_script system procedure" [*MobiLink - Server Administration*] and "ml_add_dnet_table_script system procedure" [*MobiLink - Server Administration*].

In the next section you connect to CustDB with Interactive SQL and execute ml_add_dnet_table_script, assigning MLExample.CustdbScripts.DownloadCursor to the download_cursor event.

To specify MLExample.CustdbScripts.DownloadCursor for the ULCustomer download_cursor event

1. Connect to the sample database with Interactive SQL.
   - Start Interactive SQL.

     Choose **Start** » **Programs** » **SQL Anywhere 11** » **Interactive SQL**, or run the following command:

     ```
     dbisql
     ```

     The Connect window appears.
   - On the **Identification** tab, choose the **ODBC Data Source** SQL Anywhere 11 CustDB.
   - On the **Database** tab, ensure the option to search for network database servers is not selected.
   - Click **OK** to connect.
2. Execute the following command in Interactive SQL:

   ```
   CALL ml_add_dnet_table_script(
   'custdb 11.0',
   'ULCustomer',
   'download_cursor',
   'MLExample.CustdbScripts.DownloadCursor');
   COMMIT;
   ```

   Following is a description of each parameter:

Parameter	Description
custdb 11.0	The script version.
ULCustomer	The synchronized table.
download_cursor	The event name.
MLExample.CustdbScripts.DownloadCursor	The fully qualified .NET method.
3. Exit Interactive SQL.

In this lesson, you specified .NET methods to handle ULCustomer table-level events. The next lesson ensures that the MobiLink server loads the appropriate class files and the MobiLink server API.

Further reading

For more information about adding and deleting synchronization scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

For more information about the scripts used in this lesson, see "ml_add_dnet_connection_script system procedure" [*MobiLink - Server Administration*] and "ml_add_dnet_table_script system procedure" [*MobiLink - Server Administration*].

Lesson 3: Run MobiLink with -sl dnet

Running the MobiLink server with the -sl dnet option specifies the location of .NET assemblies and forces the CLR to load on server startup.

If you compiled using Visual Studio .NET, the location of *CustdbScripts.dll* is *c:\mldnet\CustdbScripts\CustdbScripts\bin\Debug*. If you compiled at a command prompt, the location of *CustdbScripts.dll* is *c:\mldnet*.

To start the MobiLink server (mlsrv11) and load .NET assemblies

- Start the MobiLink server with the -sl dnet option.

  If you used Visual Studio .NET to compile your assembly:

  At a command prompt, type the following command on a single line:

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -dl -o cons1.txt -v+ -sl dnet(-
  MLAutoLoadPath=c:\mldnet\CustdbScripts\CustdbScripts\bin\Debug)
  ```

  If you compiled your assembly at a command prompt:

  At a command prompt, type the following command on a single line:

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -dl -o cons1.txt -v+ -sl dnet(-
  MLAutoLoadPath=c:\mldnet)
  ```

  A message window appears indicating that the server is ready to handle requests. Now the .NET method is executed when the upload_insert events triggers during synchronization.

Further reading

For more information, see "-sl dnet option" [*MobiLink - Server Administration*].

Lesson 4: Test synchronization

UltraLite comes with a sample Windows client that automatically invokes the dbmlsync utility when the user issues a synchronization. It is a simple sales-status application that you can run against the CustDB consolidated database you started in the previous lesson.

Start the application

To start and synchronize the sample application

1. Launch the sample application.

   From the **Start** menu, choose **Programs** » **SQL Anywhere 11** » **UltraLite** » **Windows Sample Application**.

2. Enter an employee ID.

   Input a value of **50** and press Enter.

The application automatically synchronizes, and a set of customers, products, and orders are downloaded to the application from the CustDB consolidated database.

In the next section you enter a new customer name and order details. During a subsequent synchronization, this information is uploaded to the CustDB consolidated database and the upload_insert and download_cursor events for the ULCustomer table triggers.

Add an order

To add an order

1. From the **Order** menu, choose **New**.

   The **Add New Order** window appears.

2. Enter a new Customer Name.

   For example, enter **Frank DotNET**.

3. Choose a product, and enter the quantity and discount.

4. Press Enter to add the new order.

You have now modified the data in your local UltraLite database. This data is not shared with the consolidated database until you synchronize.

To synchronize with the consolidated database and trigger the upload_insert event

- From the **File** menu, choose **Synchronize**.

  A window appears indicating that one Insert successfully uploaded to the consolidated database.

Further reading

For more information about the CustDB Windows Application, see "Exploring the CustDB sample for MobiLink" on page 57.

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Return the ULCustomer table upload_insert and download_cursor scripts to their original SQL logic.
   - Open Interactive SQL.

     Choose **Start** » **Programs** » **SQL Anywhere 11** » **Interactive SQL**, or run the following command:

     ```
     dbisql
     ```

     The **Connect** window appears.
   - On the **Identification** tab, choose the **ODBC Data Source** SQL Anywhere 11 CustDB.
   - Click **OK** to connect.
   - Execute the following commands in Interactive SQL:

     ```
     CALL ml_add_table_script( 'custdb 11.0',
        'ULCustomer',
        'upload_insert',
        'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

     CALL ml_add_table_script( 'custdb 11.0',
        'ULCustomer',
        'download_cursor',
        'SELECT "cust_id", "cust_name"
         FROM "ULCustomer"
         WHERE "last_modified" >= ?' );
     ```
2. Close the SQL Anywhere, MobiLink, and synchronization client windows by right-clicking each taskbar item and choosing **Close**.
3. Delete all tutorial-related .NET sources.

   Delete the folder containing your *CustdbScripts.cs* and *CustdbScripts.dll* files (*c:\mldnet*).

> **Note**
> Ensure that you do not have other important files in *c:\mldnet*.

Further reading

For more information about writing MobiLink synchronization scripts in .NET, see "Setting up .NET synchronization logic" [*MobiLink - Server Administration*].

For information about debugging .NET synchronization logic, see "Debugging .NET synchronization logic" [*MobiLink - Server Administration*].

For a detailed example illustrating the use of .NET synchronization scripts for custom authentication, see ".NET synchronization example" [*MobiLink - Server Administration*].

For more information about synchronization scripting, see "Writing synchronization scripts" [*MobiLink - Server Administration*] and "Synchronization events" [*MobiLink - Server Administration*].

For an introduction to other methods of synchronization such as timestamp, see "Synchronization techniques" [*MobiLink - Server Administration*].

CHAPTER 10

Tutorial: Using .NET and Java for custom authentication

Contents

Introduction to MobiLink custom authentication

MobiLink synchronization scripts can be written in SQL, Java, or .NET. You can use Java or .NET to add custom actions at any point of a synchronization.

In this tutorial, you add a Java or .NET method for the authenticate_user connection event. The authenticate_user event allows you to specify a custom authentication scheme and override the MobiLink built-in client authentication.

Required software

- SQL Anywhere 11
- Java Software Development Kit

Competencies and experience

You require:

- familiarity with Java
- basic knowledge of MobiLink event scripts

Goals

You gain competence and familiarity with:

- MobiLink custom authentication

Key concepts

This section uses the following steps to implement basic Java-based synchronization using the MobiLink CustDB sample database:

- compiling a source file with MobiLink server API references
- specifying class methods for particular table-level events
- running the MobiLink server (mlsrv11) with the -sl java option
- testing synchronization with a sample Windows client application

Suggested background reading

For more information about authenticating MobiLink clients, see "Choosing a user authentication mechanism" [*MobiLink - Client Administration*].

For more information about integrating POP3, IMAP, or LDAP authentication, see "Authenticating to external servers" [*MobiLink - Client Administration*].

For more information about .NET or Java synchronization scripts, see "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*] or "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

Lesson 1: Create a Java or .NET class for custom authentication (server-side)

In this lesson, you compile a class containing Java or .NET logic for custom authentication.

MobiLink server API for .NET

To execute .NET synchronization logic, the MobiLink server must have access to the classes in *iAnywhere.MobiLink.Script.dll*. *iAnywhere.MobiLink.Script.dll* contains a repository of MobiLink .NET server API classes to utilize in your .NET methods. When you compile your .NET class, you reference *iAnywhere.MobiLink.Script.dll*.

MobiLink server API for Java

To execute Java synchronization logic, the MobiLink server must have access to the classes in *mlscript.jar*. *mlscript.jar* contains a repository of MobiLink Java server API classes to utilize in your Java methods. When you compile your Java class, you reference *mlscript.jar*.

To create your Java or .NET class for custom authentication

1. Create a class called MobiLinkAuth using Java or .NET.

   The MobiLinkAuth class includes the authenticateUser method used for the authenticate_user synchronization event. The authenticate_user event provides parameters for the user and password. You return the authentication result in the authentication_status inout parameter.

   For Java, type the following:

```
import ianywhere.ml.script.*;

public class MobiLinkAuth
{

 public void authenticateUser (
  ianywhere.ml.script.InOutInteger authentication_status,
  String user,
  String pwd,
  String newPwd )
 {
   // to do...

 }

}
```

   For .NET, type the following:

```
using iAnywhere.MobiLink.Script;

public class MobiLinkAuth
{
  public void authenticateUser  (
    ref int authentication_status,
    string user,
    string pwd,
    string newPwd)
  {
```

```
    // to do...

  }
}
```

2. Write the authenticateUser method.

   This tutorial illustrates a very simple case of custom user authentication. Authentication succeeds if the user name starts with "ML".

   Note
   You register the authenticateUser method for the authenticate_user synchronization event in "Lesson 2: Register your Java or .NET scripts for the authenticate_user event" on page 160.

   For Java, type the following:

```
public void authenticateUser (
 ianywhere.ml.script.InOutInteger authentication_status,
 String user,
 String pwd,
 String newPwd ) {

if(user.substring(0,1)=="ML") {
        // success: an auth status code of 1000
        authentication_status.setValue(1000);
    } else {
        // fail: an authentication_status code of 4000
        authentication_status.setValue(4000);
    }
}
```

   For .NET, type the following:

```
public void authenticateUser(
 ref int authentication_status,
 string user,
 string pwd,
 string newPwd ) {

    if(user.Substring(0,2)=="ML") {
        // success: an auth status code of 1000
        authentication_status = 1000;
    } else {
        // fail: and authentication_status code of 4000
        authentication_status = 4000;
    }
}
```

3. Compile the MobiLinkAuth class.

   - For Java, save the file as *MobiLinkAuth.java* in *c:\mlauth*. Navigate to *c:\mlauth*. To compile the file, run the following command. (Replace *install-dir* with the location of your SQL Anywhere installation.)

     ```
     javac MobiLinkAuth.java -classpath "install-dir\java\mlscript.jar"
     ```

     The *MobiLinkAuth.class* file is generated.

   - For .NET, save the file as *MobiLinkAuth.cs* in *c:\mlauth*.

- At a command prompt, navigate to *c:\mlauth*. To compile the file, run the following command. (Replace *install-dir* with the location of your SQL Anywhere installation.)

  ```
  csc /out:c:\mlauth\MobiLinkAuth.dll /target:library /
  reference:"%SQLANY11%\Assembly\v2\iAnywhere.MobiLink.Script.dll"
  MobiLinkAuth.cs
  ```

  The *MobiLinkAuth.dll* assembly is generated.

Further reading

For more information about the authenticate_user event, including a table of authentication_status return codes, see "authenticate_user connection event" [*MobiLink - Server Administration*].

For more information about implementing custom authentication using Java or .NET, see "Java and .NET user authentication" [*MobiLink - Client Administration*].

For a detailed example of Java custom authentication, see "Java synchronization example" [*MobiLink - Server Administration*].

For a detailed example of .NET custom authentication, see ".NET synchronization example" [*MobiLink - Server Administration*].

Lesson 2: Register your Java or .NET scripts for the authenticate_user event

In this lesson, you register the MobiLinkAuth authenticateUser method for the authenticate_user synchronization event. You add this script to CustDB, the MobiLink sample database.

MobiLink database sample

SQL Anywhere ships with a SQL Anywhere sample database (CustDB) that is already set up for synchronization. The CustDB ULCustomer table, for example, is a synchronized table supporting a variety of table-level scripts.

CustDB is designed to be a consolidated database server for both UltraLite and SQL Anywhere clients. The CustDB database has a DSN called SQL Anywhere 11 CustDB.

To register the authenticateUser method for the authenticate_user event

1. Connect to the sample database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "dsn=SQL Anywhere 11 CustDB"
   ```

2. Use the ml_add_java_connection_script or ml_add_dnet_connection_script stored procedure to register the authenticateUser method for the authenticate_user event.

   For Java, execute the following command in Interactive SQL:

   ```
   call ml_add_java_connection_script(
   'custdb 11.0',
   'authenticate_user',
   'MobiLinkAuth.authenticateUser');
   commit;
   ```

   For .NET, execute the following command in Interactive SQL:

   ```
   call ml_add_dnet_connection_script(
   'custdb 11.0',
   'authenticate_user',
   'MobiLinkAuth.authenticateUser');
   commit;
   ```

In the next lesson, you start the MobiLink server and load your class file or assembly.

Further reading

For general information about adding and deleting synchronization scripts, see “Adding and deleting scripts” [*MobiLink - Server Administration*].

For more information about the ml_add_java_connection_script, see “ml_add_java_connection_script system procedure” [*MobiLink - Server Administration*].

For more information about the ml_add_dnet_connection_script, see “ml_add_dnet_connection_script system procedure” [*MobiLink - Server Administration*].

Lesson 3: Start the MobiLink server for Java or .NET

Starting the MobiLink server with the -sl java or -sl dnet option allows you to specify a set of directories to search for compiled files.

To start the MobiLink server (mlsrv11)

- Connect to the MobiLink sample database and load your Java class or .NET assembly on the mlsrv11 command line.

  For Java, run the following command:

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -sl java(-cp c:\mlauth)
  ```

  For .NET, run the following command:

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -sl dnet(-MLAutoLoadPath=c:\mlauth)
  ```

  The MobiLink server window appears. Now, your Java or .NET method is executed when the authenticate_user synchronization event occurs.

Further reading

For more information about starting the MobiLink server for Java, see "-sl java option" [*MobiLink - Server Administration*].

For more information about starting the MobiLink server for .NET, see "-sl dnet option" [*MobiLink - Server Administration*].

Lesson 4: Test the authentication

UltraLite comes with a sample Windows client that automatically invokes the dbmlsync utility when the user issues a synchronization. It is a simple sales-status application that you can run against the CustDB consolidated database you started in the previous lesson.

To start the sample application and test authentication

1. Start the sample application.

   From the **Start** menu, choose **Programs** » **SQL Anywhere 11** » **UltraLite** » **Windows Sample Application**.

2. Enter an invalid employee ID and synchronize.

   In this application, the employee ID is also the MobiLink user name. If the user name does not begin with "ML", your Java or .NET logic causes synchronization to fail. Input a value of **50** for the employee ID and press Enter.

   The UltraLite CustDB Demo window appears indicating a synchronization error with SQL Code -103. SQL Code -103 represents an invalid user ID or password.

Further reading

For more information about the CustDB Windows Application, see “Exploring the CustDB sample for MobiLink” on page 57.

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Remove the authenticate_user script from the consolidated database.
   - Connect to the MobiLink sample database in Interactive SQL.

     Run the following command:

     ```
     dbisql -c "dsn=SQL Anywhere 11 CustDB"
     ```
   - Remove the authenticate_user script.

     For Java, run the following command to remove the authenticate_user script:

     ```
     call ml_add_java_connection_script(
     'custdb 11.0',
     'authenticate_user',
     null);
     commit;
     ```

     For .NET, run the following command to remove the authenticate_user script:

     ```
     call ml_add_dnet_connection_script(
     'custdb 11.0',
     'authenticate_user',
     null);
     commit;
     ```
2. Delete your Java or .NET source files.

   For example, delete the *c:\mlauth* directory.

   Caution
   Make sure you only have tutorial related materials in this directory.
3. Close Interactive SQL and UltraLite Windows client application.

   Choose **Exit** from the **File** menu of each application.
4. Close the SQL Anywhere, MobiLink, and synchronization client windows.

   Right-click each task-bar item and choose **Close**.

Further reading

For more information about Java synchronization logic, see "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

For more information about .NET synchronization logic, see "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*].

For a detailed example illustrating the use of Java or .NET synchronization scripts for custom authentication, see "Java synchronization example" [*MobiLink - Server Administration*] or ".NET synchronization example" [*MobiLink - Server Administration*], respectively.

For information about debugging Java or .NET synchronization logic, see "Debugging Java classes" [*MobiLink - Server Administration*] or "Debugging .NET synchronization logic" [*MobiLink - Server Administration*], respectively.

For more information about synchronization scripts, see "Writing synchronization scripts" [*MobiLink - Server Administration*] and "Synchronization events" [*MobiLink - Server Administration*].

CHAPTER 11

Tutorial: Introduction to direct row handling

Contents

Introduction to direct row handling tutorial

This tutorial shows you how to implement MobiLink direct row handling so that you can use a data source other than a supported consolidated database.

You can use direct row handling to communicate remote data to any central data source, application, or web service.

This tutorial shows you how to use the MobiLink server APIs for Java and .NET for simple direct row handling. You synchronize the client RemoteOrders table with the consolidated database and add special direct row handling processing for the OrderComments table.

You set up a simple synchronization for the RemoteOrders table.

Required software

- Java Software Development Kit or the Microsoft .NET Framework

Competencies and experience

You require:

- Familiarity with Java or .NET.
- Basic knowledge of MobiLink event scripts and MobiLink synchronization.

Goals

You gain competence and familiarity with:

- The MobiLink server APIs for Java and .NET.
- Creating methods for MobiLink direct row handling.

Suggested background reading

- "Introducing MobiLink synchronization" on page 3
- "Synchronization techniques" [*MobiLink - Server Administration*]
- "Direct row handling" [*MobiLink - Server Administration*]

MobiLink code exchange samples are located at http://ianywhere.codexchange.sybase.com/servlets/ProjectDocumentList?folderID=319.

You can post questions on the MobiLink newsgroup: sybase.public.sqlanywhere.mobilink

Lesson 1: Set up your MobiLink consolidated database

Your MobiLink consolidated database is a central repository of data and includes MobiLink system tables and stored procedures used to manage the synchronization process. With direct row handling, you synchronize with a data source other than a consolidated database, but you still need a consolidated database to maintain information used by the MobiLink server.

In this lesson, you:

- Create a database and define an ODBC data source.
- Add data tables to synchronize to remote clients.
- Install MobiLink system tables and stored procedures.

> **Note**
> If you already have a MobiLink consolidated database set up with MobiLink system objects and a DSN, you can skip this lesson.

Create your consolidated database

In this tutorial, you create a SQL Anywhere database using the Sybase Central **Create Database Wizard**.

To create your SQL Anywhere RDBMS

1. Start Sybase Central.

   Choose **Start** » **Programs** » **SQL Anywhere 11** » **Sybase Central**.

   Sybase Central appears.
2. In Sybase Central, choose **Tools** » **SQL Anywhere 11** » **Create Database**.

   The **Create Database Wizard** appears.
3. Click **Next**.
4. Leave the default setting, **Create Database On This Computer**, and then click **Next**.
5. Enter the file name and path for the database. For example, enter:

   c:\MLdirect\MLconsolidated.db
6. Follow the remaining instructions in the wizard, accepting the default values, except as follows:
   - Clear the option to **Stop The Database After Last Disconnect**.
7. Click **Finish**.

   The database called MLconsolidated appears in Sybase Central.

Define an ODBC data source for the consolidated database.

Use the SQL Anywhere 11 driver to define an ODBC data source for the MLconsolidated database.

To define an ODBC data source for the consolidated database

1. Start the ODBC Administrator:

   From the Sybase Central **Tools** menu, choose **SQL Anywhere 11** » **Open ODBC Administrator**.

   The **ODBC Data Source Administrator** appears.
2. On the **User DSN** tab, click **Add**.

   The **Create New Data Source** window appears.
3. Select **SQL Anywhere 11**, and then click **Finish**.

   The **ODBC Configuration For SQL Anywhere 11** window appears.
4. On the **ODBC** tab, type the **Data source Name mobilink_db**. On the **Login** tab, type **DBA** for the **User ID** and **sql** for the **Password**. On the **Database** tab, type **MLconsolidated** for the **Server Name** and *c:\MLdirect\MLconsolidated.db* for the **Database File**.
5. Click **OK** to define the data source and **OK** again to close the **ODBC Administrator** window.

Create tables for synchronization

In this procedure, you create the RemoteOrders table in the MobiLink consolidated database. The RemoteOrders table contains the following columns:

Column	Description
order_id	A unique identifier for orders.
product_id	A unique identifier for products.
quantity	The number of items sold.
order_status	The order status.
last_modified	The last modification date of a row. You use this column for timestamp based downloads, a common technique used to filter rows for efficient synchronization.

To create the RemoteOrders table

1. Connect to your database using Interactive SQL.

   At a command prompt, run the following command:

   ```
   dbisql -c "dsn=mobilink_db"
   ```
2. Run the following command in Interactive SQL to create the RemoteOrders table.

   ```
   CREATE TABLE RemoteOrders
   (
    order_id           integer not null,
    product_id         integer not null,
    quantity           integer,
    order_status       varchar(10) default 'new',
    last_modified      timestamp default current timestamp,
    primary key(order_id)
   )
   ```


Stay connected in Interactive SQL for the following procedure.

Run MobiLink setup scripts

You can find setup scripts for each supported consolidated database in the *MobiLink/setup* subdirectory of your SQL Anywhere 11 installation.

In this procedure you set up a SQL Anywhere consolidated database. You do this using the *syncsa.sql* setup script. Running *syncsa.sql* creates a series of system tables and stored procedures prefaced with **ml_**. The MobiLink server works with these tables and stored procedures in the synchronization process.

To install MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. Run the following command in Interactive SQL to create MobiLink system tables and stored procedures. Replace *c:\Program Files\SQL Anywhere 11* with the location of your SQL Anywhere 11 installation.

   ```
   read "c:\Program Files\SQL Anywhere 11\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL applies *syncsa.sql* to your consolidated database.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about creating SQL Anywhere databases, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about creating tables, see "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*].

For information about setting up MobiLink consolidated databases, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

Lesson 2: Add synchronization scripts

In this lesson you add scripts to your consolidated database for SQL row handling and direct row handling.

SQL row handling

SQL row handling allows you to synchronize remote data with tables in your MobiLink consolidated database. The SQL-based scripts define:

- How data that is uploaded from a MobiLink client is to be applied to the consolidated database.
- What data should be downloaded from the consolidated database.

In this lesson, you write synchronization scripts for the following SQL-based upload and download events.

- **upload_insert** To define how new orders inserted in a remote client database should be applied to the consolidated database.
- **download_cursor** To define what orders updated in the MobiLink consolidated database should be downloaded to remote clients.

In this procedure, you add synchronization script information to your MobiLink consolidated database using stored procedures.

To add SQL-based scripts to MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. Use the ml_add_table_script stored procedure to add SQL-based table scripts for the upload_insert and download_cursor events.

   Run the following commands in Interactive SQL. The upload_insert script inserts the uploaded order_id, product_id, quantity, and order_status into the MobiLink consolidated database. The download_cursor script uses timestamp-based filtering to download updated rows to remote clients.

   ```
   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'upload_insert',
        'INSERT INTO RemoteOrders( order_id, product_id, quantity,
   order_status)
         VALUES( ?, ?, ?, ? )' );

   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'download_cursor',
        'SELECT order_id, product_id, quantity, order_status
         FROM RemoteOrders WHERE last_modified >= ?');

   commit
   ```

Direct row handling processing

In this tutorial, you use direct row handling to add special processing to a SQL-based synchronization system. In this procedure you register method names corresponding to the handle_UploadData, handle_DownloadData, and end_download events. You create your own Java or .NET class in “Lesson 3: Write Java or .NET logic for processing direct row handling” on page 173.

To add information for direct row handling in MobiLink system tables

1. Connect to your consolidated database in Interactive SQL.

   Run the following command:

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. Register a Java or .NET method for the end_download event.

   You use this method to free i/o resources when the end_download connection event fires.

   For Java, execute the following command in Interactive SQL.

   ```
   CALL ml_add_java_connection_script( 'default',
      'end_download',
        'MobiLinkOrders.EndDownload' );
   ```

   For .NET, execute the following command in Interactive SQL.

   ```
   CALL ml_add_dnet_connection_script( 'default',
      'end_download',
        'MobiLinkOrders.EndDownload' );
   ```

   Interactive SQL registers the user-defined EndDownload method for the end_download event.

3. Register Java or .NET methods for the handle_UploadData and handle_DownloadData synchronization events.

   For Java, execute the following commands in Interactive SQL.

   ```
   CALL ml_add_java_connection_script( 'default',
      'handle_UploadData',
        'MobiLinkOrders.GetUpload' );

   CALL ml_add_java_connection_script( 'default',
      'handle_DownloadData',
        'MobiLinkOrders.SetDownload' );

   commit
   ```

   For .NET, execute the following commands in Interactive SQL.

   ```
   CALL ml_add_dnet_connection_script( 'default',
      'handle_UploadData',
        'MobiLinkOrders.GetUpload' );

   CALL ml_add_dnet_connection_script( 'default',
      'handle_DownloadData',
        'MobiLinkOrders.SetDownload' );

   commit
   ```

   Interactive SQL registers the user-defined GetUpload and SetDownload methods for the handle_UploadData and handle_DownloadData events, respectively.

Further reading

For information about using SQL-based events to upload data from remote clients to a MobiLink consolidated database, see:

- "Writing scripts to upload rows" [*MobiLink - Server Administration*]
- "upload_insert table event" [*MobiLink - Server Administration*]
- "upload_update table event" [*MobiLink - Server Administration*]
- "upload_delete table event" [*MobiLink - Server Administration*]

For information about uploading data to data sources other than consolidated databases, see "Handling direct uploads" [*MobiLink - Server Administration*].

For information about using SQL-based events to download data from a MobiLink consolidated database, see:

- "Writing scripts to download rows" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]
- "download_delete_cursor table event" [*MobiLink - Server Administration*]

For information about downloading data to data sources other than consolidated databases, see "Handling direct downloads" [*MobiLink - Server Administration*].

For information about the synchronization event sequence, see "Overview of MobiLink events" [*MobiLink - Server Administration*].

For information about synchronization techniques for download filtering, see "Timestamp-based downloads" [*MobiLink - Server Administration*] and "Partitioning rows among remote databases" [*MobiLink - Server Administration*].

For information about managing scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

For information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Lesson 3: Write Java or .NET logic for processing direct row handling

In this lesson, you use direct row handling to process rows in the OrderComments table in your client database. You add the following methods for direct row handling:

- **GetUpload** You use this method for the handle_UploadData event. GetUpload writes uploaded comments to a file called *orderComments.txt*.
- **SetDownload** You use this method for the handle_DownloadData event. Set download uses the *orderResponses.txt* file to download responses to remote clients.

You also add the EndDownload method to handle the end_download event.

The following procedure shows you how to create a Java or .NET class including your methods for processing. For a complete listing, see "Complete MobiLinkOrders listing (Java)" on page 179 or "Complete MobiLinkOrders listing (.NET)" on page 181.

To create a Java or .NET class for direct row handling

1. Create a class called MobiLinkOrders using Java or .NET.

   For Java, type the following code in a text editor or development environment.

   ```
   import ianywhere.ml.script.*;
   import java.io.*;
   import java.sql.*;

   public class  MobiLinkOrders{

   // to do...
   ```

   For .NET, use the following code:

   ```
   using iAnywhere.MobiLink.Script;
   using System.IO;
   using System.Data;
   using System.Text;

   public class MobiLinkOrders

   // to do...
   ```

2. Declare a class-level DBConnectionContext instance.

   In Java:

   ```
   // class level DBConnectionContext
       DBConnectionContext _cc;
   ```

   In .NET:

   ```
   // class level DBConnectionContext
       private DBConnectionContext _cc = null;
   ```

The MobiLink server passes a DBConnectionContext instance to your class constructor. DBConnectionContext encapsulates information about the current connection with the MobiLink consolidated database.

3. Create your class constructor.

   Your class constructor sets your class-level DBConnectionContext instance.

   For Java:

   ```
   public MobiLinkOrders( DBConnectionContext cc )
   {
    // set your class-level DBConnectionContext
    _cc = cc;
   }
   ```

   For .NET:

   ```
   public MobiLinkOrders( DBConnectionContext cc )
   {
    _cc = cc;
   }
   ```

4. Declare objects used for file input and output.

   For Java, declare a java.io.FileWriter and java.io.BufferedReader:

   ```
   // java objects for file i/o
       FileWriter my_writer;
       BufferedReader my_reader;
   ```

   For .NET, declare a Stream Writer and Stream Reader:

   ```
   // instances for file I/O
   private static StreamWriter   my_writer = null;
   private static StreamReader   my_reader = null;
   ```

5. Write the EndDownload method.

   This method handles the end_download connection event and gives you an opportunity to free resources.

   For Java:

   ```
   public void EndDownload() throws IOException
   {
     // free i/o resources
       if (my_reader!=null) my_reader.close();
       if (my_writer!=null) my_writer.close();
   }
   ```

   For .NET:

   ```
   public void EndDownload()
   {
     if( my_writer != null ) {
        my_writer.Close();
        my_writer = null;
     }
   ```

6. Write the GetUpload method

   The GetUpload method obtains an UploadedTableData class instance representing the OrderComments table. The OrderComments table contains special comments made by remote sales employees. You create

this table in "Lesson 5: Set up your MobiLink client" on page 185. The UploadedTableData getInserts method returns a result set for new order comments. The writeOrderComment method writes out each row in the result set to a text file.

For Java:

```
//method for the handle_UploadData synchronization event
public void GetUpload( UploadData ut ) throws SQLException, IOException
    {
   // get an UploadedTableData for OrderComments
   UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");

   // get inserts uploaded by the MobiLink client
   ResultSet insertResultSet = orderCommentsTbl.getInserts();

   while( insertResultSet.next() ) {

      // get order comments
      int _commentID = insertResultSet.getInt("comment_id");
      int _orderID = insertResultSet.getInt("order_id");
      String _specialComments =
insertResultSet.getString("order_comment");

      if (_specialComments != null)
      {
         writeOrderComment(_commentID,_orderID,_specialComments);

      }
              }
   insertResultSet.close();

   }

// writes out comment details to file
public void writeOrderComment( int _commentID, int _orderID, String
_comments )
   throws IOException
    {

      // a FileWriter for writing order comments
      if(my_writer == null)
      {
        my_writer = new FileWriter( "C:\\MLdirect\
\orderComments.txt",true);
      }

      // write out the order comments to remoteOrderComments.txt
      my_writer.write(_commentID + "\t" + _orderID + "\t" + _comments);
      my_writer.write( "\n" );
      my_writer.flush();

    }
```

In .NET:

```
// method for the handle_UploadData synchronization event.
public void GetUpload( UploadData ut )
    {
 // get UploadedTableData for remote table called OrderComments
 UploadedTableData order_comments_table_data =
ut.GetUploadedTableByName( "OrderComments" );
```

```
// get inserts upload by the MobiLink client
IDataReader new_comment_reader = order_comments_table_data.GetInserts();

while( new_comment_reader.Read() ) {
    // columns are
    // 0 - "order_comment"
    // 1 - "comment_id"
    // 2 - "order_id"
    // you can look up these values using the DataTable returned by:
    // order_comments_table_data.GetSchemaTable() if the send column
names
    // option is turned on the remote.
    // in this example you just use the known column order to determine
the column
    // indexes

    // only process this insert of the order_comment is not null
    if( !new_comment_reader.IsDBNull( 2 ) ) {
  int comment_id = new_comment_reader.GetInt32( 0 );
  int order_id   = new_comment_reader.GetInt32( 1 );
  string comments= new_comment_reader.GetString( 2 );
  WriteOrderComment( comment_id, order_id, comments );
    }
}
// always close the reader when you are done with it!
new_comment_reader.Close();
   }
```

7. Write the SetDownload method:

   a. Obtain a class instance representing the OrderComments table.

      Use the DBConnectionContext getDownloadData method to obtain a DownloadData instance. Use the DownloadData getDownloadTableByName method to return a DownloadTableData instance for the OrderComments table.

      For Java:

      ```
      DownloadData download_d = _cc.getDownloadData();
      DownloadTableData download_td =
      download_d.getDownloadTableByName( "OrderComments" );
      ```

      For .NET:

      ```
      DownloadTableData comments_for_download =
          _cc.GetDownloadData().GetDownloadTableByName( "OrderComments" );
      ```

      Note
      You create this table on the remote database in "Lesson 5: Set up your MobiLink client" on page 185.

   b. Obtain a prepared statement or IDBCommand that allows you to add insert or update operations to the download.

      For Java, Use the DownloadTableData getUpsertPreparedStatement method to return a java.sql.PreparedStatement instance.

      ```
      PreparedStatement update_ps = download_td.getUpsertPreparedStatement();
      ```

      For .NET, use the DownloadTableData GetUpsertCommand method:


```
// add upserts to the set of operation that are going to be applied at
the
// remote database
IDbCommand  comments_upsert = comments_for_download.GetUpsertCommand();
```

c. Send down responses to remote clients.

   Create a text file called *orderResponses.txt* in *c:\MLdirect*. This file contains responses to comments. For example, *orderResponses.txt* can contain the following entries including tab-delimited values representing the comment_id, order_id, and order_comment.

```
...
786  34  OK, we will ship promotional material.
787  35  Yes, the product is going out of production.
788  36  No, we can't increase your commission...
...
```

d. Set the download data for each row.

   For Java, the following example traverses through the *orderResponses.txt* and adds data to the MobiLink download.

   For Java:

```
//   a BufferedReader for reading in responses
if (my_reader==null)
 my_reader = new BufferedReader(new FileReader( "c:\\MLdirect\
\orderResponses.txt"));

// send updated comments down to clients
String commentLine;

// read the first line from orderResponses.txt
commentLine = my_reader.readLine();

   while(commentLine != null)
{
   // get the next line from orderResponses.txt
   String[] response_details = commentLine.split("\t");

   if (response_details.length != 3)
         {
            System.err.println("Error reading from
orderResponses.txt");
            System.err.println("Error setting direct row handling
download");
            return;
         }

   int comment_id = Integer.parseInt(response_details[0]);
   int order_id = Integer.parseInt(response_details[0]);
   String updated_comment = response_details[2];

   // set an order comment response in the MobiLink download
update_ps.setInt(1, comment_id);
   update_ps.setInt(2, order_id);
   update_ps.setString(3, updated_comment);
    update_ps.executeUpdate();

   // get next line from orderResponses.txt
         commentLine  = my_reader.readLine();
      }
```

   For .NET:

```
string comment_line;
while( (comment_line = my_reader.ReadLine()) != null) {
     // three values are on each line separated by '\t'
     string[] response_details = comment_line.Split( '\t' );
     if( response_details.Length != 3 ) {
throw( new SynchronizationException( "Error reading from
orderResponses.txt" ) );
     }
     int comment_id = System.Int32.Parse( response_details[0] );
     int order_id   = System.Int32.Parse( response_details[1] );
     string comments= response_details[2];

     // Parameters of the correct number and type have already been
added
     // so you just need to set the values of the IDataParameters
     ((IDataParameter)(comments_upsert.Parameters[0])).Value =
comment_id;
     ((IDataParameter)(comments_upsert.Parameters[1])).Value =
order_id;
     ((IDataParameter)(comments_upsert.Parameters[2])).Value =
comments;
     // add the upsert operation
     comments_upsert.ExecuteNonQuery();
}
```

e. Close the prepared statement used for adding insert or update operations to the download.

For Java:

```
update_ps.close();
```

For .NET, you do not need to close the IDBCommand. It is destroyed by MobiLink at the end of the download.

8. Compile your class file.
   - Navigate to the directory containing your Java or .NET source files.
   - Compile MobiLinkOrders with references to the MobiLink server API library for Java or .NET.

     For Java, you need to reference *mlscript.jar* in the *java* subdirectory of your SQL Anywhere installation. Run the following command to compile your Java class, replacing *c:\Program Files\SQL Anywhere 11* with your SQL Anywhere 11 directory:

     ```
     javac -classpath "%SQLANY11%\java\mlscript.jar" MobiLinkOrders.java
     ```

     For .NET, run the following command:

     ```
     csc /out:MobiLinkServerCode.dll /target:library /reference:"%SQLANY11%
     \Assembly\v2\iAnywhere.MobiLink.Script.dll" MobiLinkOrders.cs
     ```

Note
The example shown here does not ensure that Primary Key values are unique. See "Maintaining unique primary keys" [*MobiLink - Server Administration*].

Further reading

For more information about class constructors and DBConnectionContext, see:

- .NET synchronization logic: "Constructors" [*MobiLink - Server Administration*]

- Java synchronization logic: "Constructors" [*MobiLink - Server Administration*]

For more information about Java synchronization logic, see "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

For more information about .NET synchronization logic, see "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*].

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Complete MobiLinkOrders listing (Java)

Following is the complete MobiLinkOrders listing for Java direct row handling. For a step by step explanation, see "Lesson 3: Write Java or .NET logic for processing direct row handling" on page 173.

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

public class MobiLinkOrders {

    // class level DBConnectionContext
    DBConnectionContext _cc;

    // java objects for file i/o
    FileWriter my_writer;
    BufferedReader my_reader;
    public MobiLinkOrders( DBConnectionContext cc )
        throws IOException, FileNotFoundException
    {
        // declare a class-level DBConnectionContext
        _cc = cc;
    }

    public void writeOrderComment( int _commentID, int _orderID, String
_comments )
        throws IOException
    {
        if (my_writer == null)
            // a FileWriter for writing order comments
            my_writer = new FileWriter( "C:\\MLdirect\
\orderComments.txt",true);

        // write out the order comments to remoteOrderComments.txt
        my_writer.write(_commentID + "\t" + _orderID + "\t" + _comments);
        my_writer.write( "\n" );
        my_writer.flush();
    }

    //  method for the handle_UploadData synchronization event
    public void GetUpload( UploadData ut )
        throws SQLException, IOException
    {
        //  get an UploadedTableData for OrderComments
        UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");
```

```
        // get inserts uploaded by the MobiLink client
        ResultSet insertResultSet = orderCommentsTbl.getInserts();

        while ( insertResultSet.next() )
        {
            // get order comments
            int _commentID = insertResultSet.getInt("comment_id");
            int _orderID = insertResultSet.getInt("order_id");
            String _specialComments =
insertResultSet.getString("order_comment");
            if (_specialComments != null)
            {
                writeOrderComment(_commentID,_orderID,_specialComments);
            }
        }
        insertResultSet.close();
    }

    public void SetDownload()
        throws SQLException, IOException
    {
        DownloadData download_d = _cc.getDownloadData();

        DownloadTableData download_td =
download_d.getDownloadTableByName( "OrderComments" );

        PreparedStatement update_ps =
download_td.getUpsertPreparedStatement();
        // a BufferedReader for reading in responses
        if (my_reader == null)
            my_reader = new BufferedReader(new FileReader( "c:\\MLdirect\
\orderResponses.txt"));

        // get the next line from orderResponses
        String commentLine;
        commentLine = my_reader.readLine();

        // send comment responses down to clients
        while (commentLine != null)
        {
            // get the next line from orderResponses.txt
            String[] response_details = commentLine.split("\t");

            if (response_details.length != 3)
            {
                System.err.println("Error reading from orderResponses.txt");
                System.err.println("Error setting direct row handling
download");
                return;
            }
            int comment_id = Integer.parseInt(response_details[0]);
            int order_id = Integer.parseInt(response_details[1]);
            String updated_comment = response_details[2];

            // set an order comment response in the MobiLink download
            update_ps.setInt(1, comment_id);
            update_ps.setInt(2, order_id);
            update_ps.setString(3, updated_comment);
            update_ps.executeUpdate();

            // get next line
            commentLine = my_reader.readLine();
        }
        update_ps.close();
```



```
    }

    public void EndDownload()
        throws IOException
    {
        // close i/o resources
        if (my_reader != null) my_reader.close();
        if (my_writer != null) my_writer.close();
    }
}
```

Complete MobiLinkOrders listing (.NET)

Following is the complete MobiLinkOrders listing for .NET object-based uploads and downloads. For a step by step explanation, see "Lesson 3: Write Java or .NET logic for processing direct row handling" on page 173.

```
using iAnywhere.MobiLink.Script;
using System.IO;
using System.Data;
using System.Text;

public class MobiLinkOrders
{
    // class level DBConnectionContext
    private DBConnectionContext _cc = null;

    // instances for file I/O
    private static StreamWriter my_writer = null;
    private static StreamReader my_reader = null;


    public MobiLinkOrders(DBConnectionContext cc)
    {
        _cc = cc;
    }

    public void WriteOrderComment(int comment_id,
        int order_id,
        string comments)
    {
        if (my_writer == null)
        {
            my_writer = new StreamWriter(
                "c:\\MLdirect\\orderComments.txt");
        }
        my_writer.WriteLine("{0}\t{1}\t{2}",
            comment_id, order_id, comments);
        my_writer.Flush();
    }

    // method for the handle_UploadData synchronization event.
    public void GetUpload(UploadData ut)
    {
        // get UploadedTableData for remote table called OrderComments
        UploadedTableData order_comments_table_data =
            ut.GetUploadedTableByName("OrderComments");

        // get inserts uploaded by the MobiLink client
        IDataReader new_comment_reader =
```

```
            order_comments_table_data.GetInserts();

        while (new_comment_reader.Read())
        {
            // Columns are
            // 0 - "order_comment"
            // 1 - "comment_id"
            // 2 - "order_id"
            // You can look up these values using the DataTable returned by:
            // order_comments_table_data.GetSchemaTable() if the send
            // column names option is turned on at the remote.
            // In this example, you just use the known column order to
            // determine the column indexes

            // Only process this insert of the order_comment is not null
            if (!new_comment_reader.IsDBNull(2))
            {
                int comment_id = new_comment_reader.GetInt32(0);
                int order_id = new_comment_reader.GetInt32(1);
                string comments = new_comment_reader.GetString(2);
                WriteOrderComment(comment_id, order_id, comments);
            }
        }
        // Always close the reader when you are done with it!
        new_comment_reader.Close();
    }

    private const string read_file_path =
        "c:\\MLdirect\\orderResponses.txt";

    // method for the handle_DownloadData synchronization event
    public void SetDownload()
    {
        if ((my_reader == null) && !File.Exists(read_file_path))
        {
            System.Console.Out.Write("There is no file to read.");
            return;
        }
        DownloadTableData comments_for_download =
            _cc.GetDownloadData().GetDownloadTableByName("OrderComments");

        // Add upserts to the set of operation that are going to be
        // applied at the remote database
        IDbCommand comments_upsert =
            comments_for_download.GetUpsertCommand();

        if (my_reader == null)
        {
            my_reader = new StreamReader(read_file_path);
        }
        string comment_line;
        while ((comment_line = my_reader.ReadLine()) != null)
        {
            // three values are on each line separated by '\t'
            string[] response_details = comment_line.Split('\t');
            if (response_details.Length != 3)
            {
                throw (new SynchronizationException(
                    "Error reading from orderResponses.txt"));
            }
            int comment_id = System.Int32.Parse(response_details[0]);
            int order_id = System.Int32.Parse(response_details[1]);
            string comments = response_details[2];
```



```
                // Parameters of the correct number and type have
                // already been added so you just need to set the
                // values of the IDataParameters
                ((IDataParameter)(comments_upsert.Parameters[0])).Value =
                    comment_id;
                ((IDataParameter)(comments_upsert.Parameters[1])).Value =
                    order_id;
                ((IDataParameter)(comments_upsert.Parameters[2])).Value =
                    comments;
                // Add the upsert operation
                comments_upsert.ExecuteNonQuery();
            }
        }

        public void EndDownload()
        {
            if (my_writer != null)
            {
                my_writer.Close();
                my_writer = null;
            }
            if (my_reader != null)
            {
                my_reader.Close();
                my_reader = null;
            }
        }
    }
```

Lesson 4: Start the MobiLink server

In this lesson you start the MobiLink server. You start the MobiLink server (mlsrv11) using the -c option to connect to your consolidated database. You use the -sl java or -sl dnet option to load your Java or .NET class.

To start the MobiLink server for direct row handling

- Connect to your consolidated database and load .NET or Java classes on the mlsrv11 command line.

  For Java, run the following command. Replace *c:\MLdirect* with the location of your Java source files.

  ```
  mlsrv11 -c "dsn=mobilink_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl
  java (-cp c:\MLdirect)
  ```

  For .NET:

  ```
  mlsrv11 -c "dsn=mobilink_db;uid=DBA;pwd=sql" -o serverOut.txt -v+ -dl -zu+
  -x tcpip  -sl dnet (-Dautoloadpath=c:\MLdirect)
  ```

  The MobiLink server window appears.

  Below is a description of each MobiLink server option used in this tutorial. The options -o, -v, and -dl provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v and -dl are typically not used in production.

Option	Description
-c	Precedes the connection string.
-o	Specifies the message log file *serverOut.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.
-dl	Displays all log messages on screen.
-zu+	Adds new users automatically.
-x	Sets the communications protocol and parameters for MobiLink clients.
-sl java	Specifies a set of directories to search for class files, and forces the Java Virtual Machine to load on server startup.
-sl dnet	Specifies the location of .NET assemblies and forces the CLR to load on server startup.

Further reading

For a complete list of MobiLink server options, see "MobiLink server options" [*MobiLink - Server Administration*].

For more information about loading Java and .NET classes see "-sl java option" [*MobiLink - Server Administration*] and "-sl dnet option" [*MobiLink - Server Administration*], respectively.

Lesson 5: Set up your MobiLink client

In this tutorial, you use a SQL Anywhere database for your consolidated database and your MobiLink client. For tutorial purposes your MobiLink client, consolidated database, and MobiLink server all reside on the same computer.

To set up the MobiLink client database, you create the RemoteOrders and OrderComments tables. The RemoteOrders table corresponds to the RemoteOrders table on the consolidated database. The MobiLink server uses SQL-based scripts to synchronize remote orders. The OrderComments table is only used on client databases. The MobiLink server processes the OrderComments tables using special events.

You also create a synchronization user, publication, and subscription on the client database.

To set up your MobiLink client database

1. Create your MobiLink client database.

   In this lesson, you create a SQL Anywhere database using the dbinit command line utility.

   - To create your SQL Anywhere database, run the following command:

     ```
     dbinit -i -k remote1
     ```

     The -i and -k options tell dbinit to omit jConnect support and Watcom SQL compatibility views, respectively.

   - To start the database engine, run the following command:

     ```
     dbeng11 remote1
     ```

2. Connect to your MobiLink client database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

3. Create the RemoteOrders table.

   Execute the following command in Interactive SQL:

   ```
   create table RemoteOrders (
       order_id           integer not null,
       product_id         integer not null,
       quantity   integer,
       order_status       varchar(10) default 'new',
       primary key(order_id)
   )
   ```

4. Run the following command in Interactive SQL to create the OrderComments table:

   ```
   create table OrderComments (
       comment_id         integer not null,
       order_id           integer not null,
       order_comment      varchar (255),
       primary key(comment_id),
       foreign key (order_id) references
        RemoteOrders (order_id)
   )
   ```

5. Create your MobiLink synchronization user, publication, and subscription:

```
CREATE SYNCHRONIZATION USER ml_sales1;
CREATE PUBLICATION order_publ (TABLE RemoteOrders, Table OrderComments);
CREATE SYNCHRONIZATION SUBSCRIPTION TO order_publ FOR ml_sales1
 TYPE TCPIP ADDRESS 'host=localhost'
```

Note
You specify how to connect to the MobiLink server using the TYPE and ADDRESS clauses in the CREATE SYNCHRONIZATION SUBSCRIPTION statement.

You can use publications to determine what data is synchronized. In this case you specify the entire RemoteOrders and OrderComments tables.

Further reading

For information about creating a SQL Anywhere database, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about MobiLink clients, see "Introducing MobiLink clients" [*MobiLink - Client Administration*].

For information about creating MobiLink objects on the client, see:

- "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE PUBLICATION statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]

Lesson 6: Synchronize

The dbmlsync utility initiates MobiLink synchronization for SQL Anywhere remote databases. Before starting dbmlsync, add order data and comments to your remote database.

To set up your remote data (client-side)

1. Connect to the MobiLink client database in Interactive SQL:

   Run the following command:

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

2. Add an order to the RemoteOrders table in the client database.

   Execute the following command in Interactive SQL.

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. Add a comment to the OrderComments table in the client database.

   Execute the following command in Interactive SQL.

   ```
   INSERT INTO OrderComments (comment_id, order_id, order_comment)
    VALUES (1,1,'send promotional material with the order')
   ```

4. Commit your changes.

   Execute the following in Interactive SQL:

   ```
   COMMIT;
   ```

5. Create a blank text file called *orderResponses.txt* in the same directory as your consolidated database.

To start the synchronization client (client-side)

- At the command prompt, run the following command (on a single line):

  ```
  dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -e scn=on -o rem1.txt -v+
  ```

  Below is a description of each option:

Option	Description
-c	Specifies the connection string.
-e scn	Sets SendColumnNames to on. This is required by direct row handling if you want to reference columns by name.
-o	Specifies the message log file *rem1.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.

Once you have started the MobiLink synchronization client, an output screen appears indicating that the synchronization succeeded. SQL-based synchronization transferred rows in the client RemoteOrders table to the RemoteOrders table in the consolidated database.

Java or .NET processing inserted your comment in *orderComments.txt*. In the next procedure, you insert a response in *orderResponses.txt* to download to the remote database.

To return comments using direct row handling downloads (server-side and client-side)

1. Close any SQL Anywhere MobiLink Client windows
2. Insert return comments. This action takes place on the server side.

   Add the following text to *orderResponses.txt*. You must separate entries using the tab character. At the end of the line, press Enter.

   ```
   1  1  Promotional material shipped
   ```
3. Run synchronization using the dbmlsync client utility.

   This action takes place on the client-side.

   Run the following command:

   ```
   dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -o rem1.txt -v+ -e scn=on
   ```

   The MobiLink client utility appears.

   In Interactive SQL, select from the OrderComments table to verify that the row was downloaded.

Note
Rows downloaded using direct row handling are not printed by the mlsrv11 -v+ option, but are printed in the remote log by the remote -v+ option.

Further reading

For more information about dbmlsync, see "SQL Anywhere clients" [*MobiLink - Client Administration*].

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials

1. Close all instances of Interactive SQL.
2. Close the SQL Anywhere, MobiLink, and synchronization client windows.
3. Delete all tutorial-related DSNs:
   - Start the ODBC Administrator.

     Type the following at a command prompt:

     ```
     odbcad32
     ```
   - Remove the mobilink_db data source.
4. Delete the consolidated and remote databases:
   - Navigate to the directory containing your consolidated and remote databases.
   - Delete *MLconsolidated.db*, *MLconsolidated.log*, *remote1.db*, and *remote1.log*.

Further reading

For more information about MobiLink server APIs, see:

- "Writing synchronization scripts in Java" [*MobiLink - Server Administration*]
- "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*]

For more information about MobiLink direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

CHAPTER 12

Tutorial: Synchronizing with Microsoft Excel

Contents

Introduction to synchronizing with Excel tutorial

This tutorial guides you through the basic steps for using direct row handling to synchronize data in a Microsoft Excel spreadsheet with MobiLink clients. This tutorial uses a Java implementation as an example.

This tutorial shows you how to implement MobiLink direct row handling so that you can use a data source other than a supported consolidated database.

You can use direct row handling to communicate remote data to any central data source, application, or web service.

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

This tutorial shows you how to synchronize data in a Microsoft Excel spreadsheet to remote clients.

Required software

- SQL Anywhere 11
- Java Software Development Kit
- Microsoft Excel 2000 or later

Competencies and experience

You require:

- Familiarity with Java
- Familiarity with Microsoft Excel
- Basic knowledge of MobiLink event scripts and MobiLink synchronization

Goals

You gain competence and familiarity with:

- The MobiLink server API for Java
- Creating methods for MobiLink direct row handling
- Accessing data from a Microsoft Excel worksheet using Java

Suggested background reading

- "Introducing MobiLink synchronization" on page 3
- "Synchronization techniques" [*MobiLink - Server Administration*]
- "Direct row handling" [*MobiLink - Server Administration*]

MobiLink code exchange samples are located at http://ianywhere.codexchange.sybase.com/servlets/ProjectDocumentList?folderID=319.

You can post questions on the MobiLink newsgroup: sybase.public.sqlanywhere.mobilink

Lesson 1: Setting up your Excel worksheet

In this lesson, you create an Excel worksheet and use the Microsoft Excel Driver to define an ODBC data source. The Excel worksheet stores product information.

To set up your Excel data source

1. Create an Excel worksheet with the following contents:

order_id	comment_id	order_comment
1	2	Promotional material shipped
1	3	More information about material required

2. Change the default worksheet name **Sheet1** to **order_sheet**.
   - Double-click the **Sheet1** tab.
   - Type **order_sheet**.
3. Save the Excel workbook.

   This tutorial assumes *c:\MLobjexcel* as the working directory for server-side components. Save the workbook as *order_central.xls* in this working directory.
4. Create an ODBC data source using the Microsoft Excel Driver:
   - Start the ODBC Administrator.

     From the **Start** menu, choose **Programs** » **SQL Anywhere 11** » **ODBC Administrator**.
   - On the **User DSN** tab, click **Add**.

     The **Create New Data Source** window appears.
   - Select **Microsoft Excel Driver (*.xls)** and click **Finish**.

     The **ODBC Microsoft Excel Setup** window appears.
   - Type the **Data Source Name excel_datasource**.
   - Click **Select Workbook** and browse to *c:\MLobjexcel\order_central.xls* (the file containing your worksheet).
   - Click **OK** to create the data source. Click **OK** again to exit the ODBC Administrator.

Lesson 2: Set up your MobiLink consolidated database

Your MobiLink consolidated database is a central repository of data and includes MobiLink system tables and stored procedures used to manage the synchronization process. With direct row handling, you synchronize with a data source other than a consolidated database, but you still need a consolidated database to maintain information used by the MobiLink server.

In this lesson, you:

- Create a database and define an ODBC data source.
- Add data tables to synchronize to remote clients.
- Install MobiLink system tables and stored procedures.

> **Note**
> If you already have a MobiLink consolidated database set up with MobiLink system objects and a DSN, you can skip this lesson.

Create your consolidated database

In this tutorial, you create a SQL Anywhere database using the Sybase Central Create Database Wizard.

To create your SQL Anywhere RDBMS

1. Start Sybase Central.

   From the **Start** menu, choose **Programs** » **SQL Anywhere 11** » **Sybase Central**.

   Sybase Central appears.
2. In Sybase Central, choose **Tools** » **SQL Anywhere 11** » **Create Database**.

   The **Create Database Wizard** appears.
3. Click **Next**.
4. Leave the default of **Create Database On This Computer**, and then click **Next**.
5. Enter the file name and path for the database. For example, enter:

   c:\MLobjexcel\MLconsolidated.db
6. Follow the remaining instructions in the wizard, accepting the default values, except as follows:
   - Clear the option **Stop The Database After Last Disconnect**.
7. Click **Finish**.

   The database called MLconsolidated appears in Sybase Central.

Define an ODBC data source for the consolidated database.

Use the SQL Anywhere 11 driver to define an ODBC data source for the MLconsolidated database.

To define an ODBC data source for the consolidated database

1. Start the ODBC Administrator:

   From the Sybase Central **Tools** menu, choose **SQL Anywhere 11** » **Open ODBC Administrator**.

   The **ODBC Data Source Administrator** appears.

2. On the **User DSN** tab, click **Add**.

   The **Create New Data Source** window appears.

3. Select **SQL Anywhere 11**, and then click **Finish**.

   The **ODBC Configuration For SQL Anywhere 11** window appears.

4. On the **ODBC** tab, type the Data source name **mlexcel_db**. On the **Login** tab, type **DBA** for the **User ID** and **sql** for the **Password**. On the **Database** tab, type **MLconsolidated** for the **Server Name** and *c:\MLobjexcel\MLconsolidated.db* for the **Database File**.

5. Click **OK** to define the data source and **OK** again to close **ODBC Administrator**.

Create tables for synchronization

In this procedure, you create the RemoteOrders table in the MobiLink consolidated database. The RemoteOrders table contains the following columns:

Column	Description
order_id	A unique identifier for orders.
product_id	A unique identifier for products.
quantity	The number of items sold.
order_status	The order status.
last_modified	The last modification date of a row. You use this column for timestamp based downloads, a common technique used to filter rows for efficient synchronization.

To create the RemoteOrders table

1. Connect to your database using Interactive SQL.

   At a command prompt, run the following command:

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. Run the following command in Interactive SQL to create the RemoteOrders table.

   ```
   CREATE TABLE RemoteOrders
   (
    order_id           integer not null,
    product_id         integer not null,
    quantity           integer,
    order_status       varchar(10) default 'new',
    last_modified      timestamp default current timestamp,
    primary key(order_id)
   )
   ```

Interactive SQL creates the RemoteOrders table in your consolidated database.

Stay connected in Interactive SQL for the following procedure.

Run MobiLink setup scripts

You can find setup scripts for each supported consolidated database in the *MobiLink/setup* subdirectory of your SQL Anywhere 11 installation.

In this procedure you set up a SQL Anywhere consolidated database. You do this using the *syncsa.sql* setup script. Running *syncsa.sql* creates a series of system tables and stored procedures prefaced with **ml_**. The MobiLink server works with these tables and stored procedures in the synchronization process.

To install MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. Run the following command in Interactive SQL to create MobiLink system tables and stored procedures. Replace *c:\Program Files\SQL Anywhere 11* with the location of your SQL Anywhere 11 installation.

   ```
   read "c:\Program Files\SQL Anywhere 11\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL applies *syncsa.sql* to your consolidated database.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about creating SQL Anywhere databases, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about creating tables, see "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*].

For information about setting up MobiLink consolidated databases, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

Lesson 3: Add synchronization scripts

In this lesson you add scripts to your consolidated database for SQL row handling and direct row handling.

SQL row handling

SQL row handling allows you to synchronize remote data with tables in your MobiLink consolidated database. In this tutorial, the SQL-based scripts define:

- How data that is uploaded from a MobiLink client is to be applied to the consolidated database.
- What data should be downloaded from the consolidated database.

In this lesson, you write synchronization scripts for the following SQL-based upload and download events.

- **upload_insert** To define how new orders inserted in a remote client database should be applied to the consolidated database.
- **download_cursor** To define what orders updated in the MobiLink consolidated database should be downloaded to remote clients.

In this procedure, you add synchronization script information to your MobiLink consolidated database using stored procedures.

To add SQL-based scripts to MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. Use the ml_add_table_script stored procedure to add SQL-based table scripts for the upload_insert and download_cursor events.

   Run the following commands in Interactive SQL. The upload_insert script inserts the uploaded order_id, product_id, quantity, and order_status into the MobiLink consolidated database. The download_cursor script uses timestamp-based filtering to download updated rows to remote clients.

   ```
   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'upload_insert',
        'INSERT INTO RemoteOrders( order_id, product_id, quantity,
   order_status)
         VALUES( ?, ?, ?, ? )' );

   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'download_cursor',
        'SELECT order_id, product_id, quantity, order_status
         FROM RemoteOrders WHERE last_modified >= ?');

   commit
   ```

Direct row handling processing

In this tutorial, you use direct row handling to add special processing to a SQL-based synchronization system. In this procedure you register method names corresponding to the handle_UploadData, handle_DownloadData, and end_download events. You create your own Java class in “Lesson 4: Creating a Java class using MobiLink direct row handling” on page 200.

To add information for direct row handling in MobiLink system tables

1. Connect to your consolidated database in Interactive SQL.

   Run the following command:

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. Register Java methods for the handle_UploadData and handle_DownloadData synchronization events.

   Execute the following commands in Interactive SQL.

   ```
   CALL ml_add_java_connection_script( 'default',
      'handle_UploadData',
        'MobiLinkOrders.GetUpload' );

   CALL ml_add_java_connection_script( 'default',
      'handle_DownloadData',
        'MobiLinkOrders.SetDownload' );

   commit
   ```

   Interactive SQL registers the user-defined GetUpload and SetDownload methods for the handle_UploadData and handle_DownloadData events, respectively.

Further reading

For information about using SQL-based events to upload data from remote clients to a MobiLink consolidated database, see:

- "Writing scripts to upload rows" [*MobiLink - Server Administration*]
- "upload_insert table event" [*MobiLink - Server Administration*]
- "upload_update table event" [*MobiLink - Server Administration*]
- "upload_delete table event" [*MobiLink - Server Administration*]

For information about uploading data to data sources other than consolidated databases, see "Handling direct uploads" [*MobiLink - Server Administration*].

For information about using SQL-based events to download data from a MobiLink consolidated database, see:

- "Writing scripts to download rows" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]
- "download_delete_cursor table event" [*MobiLink - Server Administration*]

For information about downloading data to data sources other than consolidated databases, see "Handling direct downloads" [*MobiLink - Server Administration*].

For information about the synchronization event sequence, see "Overview of MobiLink events" [*MobiLink - Server Administration*].

For information about synchronization techniques for download filtering, see "Timestamp-based downloads" [*MobiLink - Server Administration*] and "Partitioning rows among remote databases" [*MobiLink - Server Administration*].

For information about managing scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

For information about direct row handling, see “Direct row handling” [*MobiLink - Server Administration*].

Lesson 4: Creating a Java class using MobiLink direct row handling

In this lesson, you use direct row handling to process rows in the OrderComments table in your client database. You add the following methods for direct row handling:

- **GetUpload** You use this method for the handle_UploadData event. GetUpload writes uploaded comments to the excel worksheet *order_central.xls*.
- **SetDownload** You use this method for the handle_DownloadData event. SetDownload retrieves the data stored in the excel worksheet *order_central.xls* and sends it to remote clients.

The following procedure shows you how to create a Java class including your methods for processing. For a complete listing, see "Complete MobiLinkOrders code listing (Java)" on page 203.

To create a Java class for download-only direct row handling

1. Create a class Called MobiLinkOrders .

   Type the following code in a text editor or development environment.

   ```
   import ianywhere.ml.script.*;
   import java.io.*;
   import java.sql.*;

   public class MobiLinkOrders {
       // ...
   }
   ```

2. Declare a class-level DBConnectionContext instance.

   ```
   DBConnectionContext _cc;
   ```

   The MobiLink server passes a DBConnectionContext instance to your class constructor. DBConnectionContext encapsulates information about the current connection with the MobiLink consolidated database.

3. Create your class constructor.

   Your class constructor sets your class-level DBConnectionContext instance.

   Type the following code in a text editor or development environment.

   ```
   public MobiLinkOrders( DBConnectionContext cc ) {
       _cc = cc;
   }
   ```

4. Write the GetUpload() method.

   The GetUpload method obtains an UploadedTableData class instance representing the OrderComments table. The OrderComments table contains special comments made by remote sales employees. You create this table in "Lesson 6: Set up your MobiLink client" on page 206. The UploadedTableData getInserts method returns a result set for new order comments.

   Type the following code in a text editor or development environment.

   ```
   //  method for the handle_UploadData synchronization event
   public void GetUpload( UploadData ut )
   ```



```
  throws SQLException, IOException {
  //  get an UploadedTableData for OrderComments
  UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");

  // get inserts uploaded by the MobiLink client
  ResultSet insertResultSet = orderCommentsTbl.getInserts();
  try {
    // connect to the excel worksheet through ODBC
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

    while( insertResultSet.next() ) {

      // get order comments
      int _commentID = insertResultSet.getInt("comment_id");
      int _orderID = insertResultSet.getInt("order_id");
      String _specialComments =
insertResultSet.getString("order_comment");

      // execute an insert statement to add the order comment to the
worksheet
      Statement st = con.createStatement();
      st.executeQuery( "insert into [order_sheet$]"
        + "(order_id, comment_id, order_comment) VALUES ("
        + Integer.toString(_orderID) + ", "
        + Integer.toString(_commentID) + ", '"
        + _specialComments + "')");

      st.close();
    }
    con.close();
  } catch(Exception ex) {
    System.err.print("Exception: ");
    System.err.println(ex.getMessage());
  }
  insertResultSet.close();
}
```

5. Write the SetDownload method:

   a. Obtain a class instance representing the OrderComments table.

      Use the DBConnectionContext getDownloadData method to obtain a DownloadData instance. Use the DownloadData getDownloadTableByName method to return a DownloadTableData instance for the OrderComments table.

      Type the following code in a text editor or development environment.

```
DownloadData download_d = _cc.getDownloadData();
DownloadTableData download_td =
download_d.getDownloadTableByName( "OrderComments" );
```

      Note
      You create this table on the remote database in "Lesson 6: Set up your MobiLink client" on page 206.

   b. Obtain a prepared statement or IDBCommand that allows you to add insert or update operations to the download.

Use the DownloadTableData getUpsertPreparedStatement method to return a java.sql.PreparedStatement instance.

Type the following code in a text editor or development environment.

```
PreparedStatement download_upserts =
download_td.getUpsertPreparedStatement();
```

c. Set the download data for each row.

The following example traverses through the *order_central.xls* worksheet and adds data to the MobiLink download.

Type the following code in a text editor or development environment.

```
try {
  // connect to the excel worksheet through ODBC
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

  // retrieve all the rows in the worksheet
  Statement st = con.createStatement();
  ResultSet Excel_rs = st.executeQuery( "select * from [order_sheet
$]" );

  while (Excel_rs.next()) {
    // retrieve the row data
    int Excel_comment_id = Excel_rs.getInt(1);
    int Excel_order_id = Excel_rs.getInt(2);
    String Excel_comment = Excel_rs.getString(3);

    // add the Excel data to the MobiLink download.
    download_upserts.setInt( 1, Excel_comment_id );
    download_upserts.setInt( 2, Excel_order_id );
    download_upserts.setString( 3, Excel_comment );
    download_upserts.executeUpdate();
  }
  // close the excel result set, statement, and connection.
  Excel_rs.close();
  st.close();
  con.close();
} catch(Exception ex) {
  System.err.print("Exception: ");
  System.err.println(ex.getMessage());
}
```

d. Close the prepared statement used for adding insert or update operations to the download.

Type the following code in a text editor or development environment.

```
download_upserts.close();
```

6. Save your Java code as *MobiLinkOrders.java* in your working directory *c:\MLobjexcel*.
7. Compile your class file.
   - Navigate to the directory containing your Java source files.
   - Compile MobiLinkOrders with references to the MobiLink server API library for Java.

     You need to reference *mlscript.jar* in the *Java* subdirectory of your SQL Anywhere installation. Run the following command to compile your Java class, replacing *c:\Program Files\SQL Anywhere 11* with your SQL Anywhere 11 directory:


```
javac -classpath "c:\Program Files\SQL Anywhere 11\java\mlscript.jar"
MobiLinkOrders.java
```

Further reading

For more information about synchronization logic, see "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Complete MobiLinkOrders code listing (Java)

The following listing shows the complete Java MobiLinkOrders class listing used for this tutorial.

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

public class MobiLinkOrders {

  // class level DBConnectionContext
  DBConnectionContext _cc;

  public MobiLinkOrders( DBConnectionContext cc )
    throws IOException, FileNotFoundException {
    // declare a class-level DBConnectionContext
    _cc = cc;
  }


  //  method for the handle_UploadData synchronization event
  public void GetUpload( UploadData ut )
    throws SQLException, IOException {
    //  get an UploadedTableData for OrderComments
    UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");

    // get inserts uploaded by the MobiLink client
    ResultSet insertResultSet = orderCommentsTbl.getInserts();
    try {
      // connect to the excel worksheet through ODBC
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

      while( insertResultSet.next() ) {
        // get order comments
        int _commentID = insertResultSet.getInt("comment_id");
        int _orderID = insertResultSet.getInt("order_id");
        String _specialComments = insertResultSet.getString("order_comment");

        // execute an insert statement to add the order comment to the
worksheet
        Statement st = con.createStatement();
        st.executeQuery( "insert into [order_sheet$]"
          + "(order_id, comment_id, order_comment) VALUES ("
          + Integer.toString(_orderID) + ", "
          + Integer.toString(_commentID) + ", '"
          + _specialComments + "')");
```

```
        st.close();
      }
      con.close();
    } catch(Exception ex) {
      System.err.print("Exception: ");
      System.err.println(ex.getMessage());
    }
    insertResultSet.close();
  }


  public void SetDownload() throws SQLException, IOException {

    DownloadData download_d = _cc.getDownloadData();

    DownloadTableData download_td =
download_d.getDownloadTableByName( "OrderComments" );

    // Prepared statement to compile upserts (inserts or updates).
    PreparedStatement download_upserts =
download_td.getUpsertPreparedStatement();
    try {
      // connect to the excel worksheet through ODBC
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

      // retrieve all the rows in the worksheet
      Statement st = con.createStatement();
      ResultSet Excel_rs = st.executeQuery( "select * from [order_sheet$]" );

      while (Excel_rs.next()) {
        // retrieve the row data
        int Excel_comment_id = Excel_rs.getInt(1);
        int Excel_order_id = Excel_rs.getInt(2);
        String Excel_comment = Excel_rs.getString(3);

        // add the Excel data to the MobiLink download.
        download_upserts.setInt( 1, Excel_comment_id );
        download_upserts.setInt( 2, Excel_order_id );
        download_upserts.setString( 3, Excel_comment );
        download_upserts.executeUpdate();
      }
      // close the excel result set, statement, and connection.
      Excel_rs.close();
      st.close();
      con.close();
    } catch (Exception ex) {
      System.err.print("Exception: ");
      System.err.println(ex.getMessage());
    }

    download_upserts.close();
  }

}
```


Lesson 5: Start the MobiLink server

In this lesson you start the MobiLink server. You start the MobiLink server (mlsrv11) using the -c option to connect to your consolidated database and the -sl java option to load your Java class, respectively.

To start the MobiLink server for direct row handling

- Connect to your consolidated database and load Java classes on the mlsrv11 command line.

  Run the following command. Replace *c:\MLobjexcel* with the location of your Java source files.

  ```
  mlsrv11 -c "dsn=mlexcel_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl
  java (-cp c:\MLobjexcel)
  ```

  The MobiLink server window appears.

  Below is a description of each MobiLink server option used in this tutorial. The options -o, -v, and -dl provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v and -dl are typically not used in production.

Option	Description
-c	Precedes the connection string.
-o	Specifies the message log file *serverOut.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.
-dl	Displays all log messages on screen.
-zu+	Adds new users automatically.
-x	Sets the communications protocol and parameters for MobiLink clients.
-sl java	Specifies a set of directories to search for class files, and forces the Java Virtual Machine to load on server startup.
-sl dnet	Specifies the location of .NET assemblies and forces the CLR to load on server startup.

Further reading

For a complete list of MobiLink server options, see "MobiLink server options" [*MobiLink - Server Administration*].

For more information about loading Java and .NET classes see "-sl java option" [*MobiLink - Server Administration*] and "-sl dnet option" [*MobiLink - Server Administration*], respectively.

Lesson 6: Set up your MobiLink client

In this tutorial, you use a SQL Anywhere database for your consolidated database and your MobiLink client. For tutorial purposes your MobiLink client, consolidated database, and MobiLink server all reside on the same computer.

To set up the MobiLink client database, you create the RemoteOrders and OrderComments tables. The RemoteOrders table corresponds to the RemoteOrders table on the consolidated database. The MobiLink server uses SQL-based scripts to synchronize remote orders. The OrderComments table is only used on client databases. The MobiLink server processes the OrderComments tables using special events.

You also create a synchronization user, publication, and subscription on the client database.

To set up your MobiLink client database

1. Create your MobiLink client database.

   In this lesson, you create a SQL Anywhere database using the dbinit command line utility.

   - To create your SQL Anywhere database, run the following command:

     ```
     dbinit -i -k remote1
     ```

     The -i and -k options tell dbinit to omit jConnect support and Watcom SQL compatibility views, respectively.

   - To start the database engine, run the following command:

     ```
     dbeng11 remote1
     ```

2. Connect to your MobiLink client database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

3. Create the RemoteOrders table.

   Execute the following command in Interactive SQL:

   ```
   create table RemoteOrders (
       order_id           integer not null,
       product_id         integer not null,
       quantity   integer,
       order_status       varchar(10) default 'new',
       primary key(order_id)
   )
   ```

4. Run the following command in Interactive SQL to create the OrderComments table:

   ```
   create table OrderComments (
       comment_id         integer not null,
       order_id           integer not null,
       order_comment      varchar (255),
       primary key(comment_id),
       foreign key (order_id) references
        RemoteOrders (order_id)
   )
   ```

5. Create your MobiLink synchronization user, publication, and subscription:


```
CREATE SYNCHRONIZATION USER ml_sales1;
CREATE PUBLICATION order_publ (TABLE RemoteOrders, Table OrderComments);
CREATE SYNCHRONIZATION SUBSCRIPTION TO order_publ FOR ml_sales1
 TYPE TCPIP ADDRESS 'host=localhost'
```

Note
You specify how to connect to the MobiLink server using the TYPE and ADDRESS clauses in the CREATE SYNCHRONIZATION SUBSCRIPTION statement.

You can use publications to determine what data is synchronized. In this case you specify the entire RemoteOrders and OrderComments tables.

Further reading

For information about creating a SQL Anywhere database, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about MobiLink clients, see "Introducing MobiLink clients" [*MobiLink - Client Administration*].

For information about creating MobiLink objects on the client, see:

- "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE PUBLICATION statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]

Lesson 7: Synchronize

The dbmlsync utility initiates MobiLink synchronization for SQL Anywhere remote databases. Before starting dbmlsync, add order data and comments to your remote database.

To set up your remote data (client-side)

1. Connect to the MobiLink client database in Interactive SQL:

   Run the following command:

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

2. Add an order to the RemoteOrders table in the client database.

   Execute the following command in Interactive SQL.

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. Add a comment to the OrderComments table in the client database.

   Execute the following command in Interactive SQL.

   ```
   INSERT INTO OrderComments (comment_id, order_id, order_comment)
    VALUES (1,1,'send promotional material with the order')
   ```

4. Commit your changes.

   Execute the following in Interactive SQL:

   ```
   COMMIT;
   ```

To start the synchronization client (client-side)

- At the command prompt, run the following command (on a single line):

  ```
  dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -e scn=on -o rem1.txt -v+
  ```

  Below is a description of each option:

Option	Description
-c	Specifies the connection string.
-e scn	Sets SendColumnNames to on. This is required by direct row handling if you want to reference columns by name.
-o	Specifies the message log file *rem1.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.

Once you have started the MobiLink synchronization client, an output screen appears indicating that the synchronization succeeded. SQL-based synchronization transferred rows in the client RemoteOrders table to the RemoteOrders table in the consolidated database.

Java processing inserted your comment in the *order_central.xls* worksheet. The information stored in the *order_central.xls* worksheet is downloaded to the client

In Interactive SQL, select from the OrderComments table to verify that the row was downloaded.

Note
Rows downloaded using direct row handling are not printed by the mlsrv11 -v+ option, but are printed in the remote log by the remote -v+ option.

Further reading

For more information about dbmlsync, see "SQL Anywhere clients" [*MobiLink - Client Administration*].

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Close all instances of the following applications.
   - Interactive SQL
   - Microsoft Excel
2. Delete the Excel workbook *order_central.xls*.
3. Close the SQL Anywhere, MobiLink, and synchronization client windows.
4. Delete all tutorial-related DSNs.
   - Start the ODBC Administrator.

     At a command prompt, type the following command:

     ```
     odbcad32
     ```
   - Remove the excel_datasource and mlexcel_db data sources.
5. Delete the consolidated and remote databases.
   - Navigate to the directory containing your consolidated and remote databases.
   - Delete *MLconsolidated.db*, *MLconsolidated.log*, *remote1.db*, and *remote1.log*.

Further reading

For more information about MobiLink synchronization, see “Introducing MobiLink synchronization” on page 3.

For more information about MobiLink clients, see “Introducing MobiLink clients” [*MobiLink - Client Administration*].

For more information about MobiLink direct row handling, see “Direct row handling” [*MobiLink - Server Administration*].

CHAPTER 13

Tutorial: Synchronizing with XML

Contents

Introduction to synchronizing with XML tutorial

This tutorial shows you how to implement MobiLink direct row handling so that you can use a data source other than a supported consolidated data source. This tutorial uses a Java implementation as an example.

You can use direct row handling to communicate remote data to any central data source, application, or web service.

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

This tutorial shows you how to synchronize data in an XML file to remote clients.

Required software

- SQL Anywhere 11
- Java Software Development Kit
- XML DOM library

Competencies and experience

You require:

- Familiarity with Java
- Familiarity with XML
- Familiarity with XML DOM
- Basic knowledge of MobiLink event scripts and MobiLink synchronization

Goals

You gain competence and familiarity with:

- The MobiLink server API for Java
- Creating methods for MobiLink direct row handling

Suggested background reading

- "Introducing MobiLink synchronization" on page 3
- "Synchronization techniques" [*MobiLink - Server Administration*]
- "Direct row handling" [*MobiLink - Server Administration*]

MobiLink code exchange samples are located at http://ianywhere.codexchange.sybase.com/servlets/ProjectDocumentList?folderID=319.

You can post questions on the MobiLink newsgroup: sybase.public.sqlanywhere.mobilink

For more information about using XML DOM with Java, see Java API for XML Processing (JAXP).

Lesson 1: Setting up your XML datasource

In this section, you create an XML file. The XML file stores order information.

To set up your XML datasource

1. Create an XML file with the following contents:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <orders></orders>
   ```

2. Save the XML file.

   This tutorial assumes *c:\MLobjxml* as the working directory for server-side components. Save the XML file as *order_comments.xml* in this working directory.

Lesson 2: Set up your MobiLink consolidated database

Your MobiLink consolidated database is a central repository of data and includes MobiLink system tables and stored procedures used to manage the synchronization process. With direct row handling, you synchronize with a data source other than a consolidated database, but you still need a consolidated database to maintain information used by the MobiLink server.

In this lesson, you:

- Create a database and define an ODBC data source.
- Add data tables to synchronize to remote clients.
- Install MobiLink system tables and stored procedures.

> **Note**
> If you already have a MobiLink consolidated database set up with MobiLink system objects and a DSN, you can skip this lesson.

Create your consolidated database

In this tutorial, you create a SQL Anywhere database using the Sybase Central Create Database Wizard.

To create your SQL Anywhere RDBMS

1. Start Sybase Central.

   Choose **Start** » **Programs** » **SQL Anywhere 11** » **Sybase Central**.

   Sybase Central appears.
2. In Sybase Central, choose **Tools** » **SQL Anywhere 11** » **Create Database**.

   The **Create Database Wizard** appears.
3. Click **Next**.
4. Leave the default of **Create Database On This Computer**, and then click **Next**.
5. Enter the file name and path for the database. For example, enter:

   c:\MLobjxml\MLconsolidated.db
6. Follow the remaining instructions in the wizard, accepting the default values, except as follows:
   - Clear the option to **Stop The Database After Last Disconnect**.
7. Click **Finish**.

   The database called MLconsolidated appears in Sybase Central.

Define an ODBC data source for the consolidated database.

Use the SQL Anywhere 11 driver to define an ODBC data source for the MLconsolidated database.

To define an ODBC data source for the consolidated database

1. Start the ODBC Administrator:

   From the Sybase Central **Tools** menu, choose **SQL Anywhere 11** » **Open ODBC Administrator**.

   The **ODBC Data Source Administrator** window appears.

2. On the **User DSN** tab, click **Add**.

   The **Create New Data Source** window appears.

3. Select **SQL Anywhere 11**, and then click **Finish**.

   The **ODBC Configuration For SQL Anywhere 11** window appears.

4. On the **ODBC** tab, type the **Data Source Name mlxml_db**. On the **Login** tab, type **DBA** for the **User ID** and **sql** for the **Password**. On the **Database** tab, type **MLconsolidated** for the **Server Name** and *c:\MLobjxml\MLconsolidated.db* for the **Database File**.

5. Click **OK** to define the data source and **OK** again to close the **ODBC Administrator** window.

Create tables for synchronization

In this procedure, you create the RemoteOrders table in the MobiLink consolidated database. The RemoteOrders table contains the following columns:

Column	Description
order_id	A unique identifier for orders.
product_id	A unique identifier for products.
quantity	The number of items sold.
order_status	The order status.
last_modified	The last modification date of a row. You use this column for timestamp based downloads, a common technique used to filter rows for efficient synchronization.

To create the RemoteOrders table

1. Connect to your database using Interactive SQL.

   You can start Interactive SQL from Sybase Central or from a command prompt.

   - To start Interactive SQL from Sybase Central, click your database **MLconsolidated - DBA**. From the Sybase Central **File** menu, choose **Open Interactive SQL**.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=mlxml_db"
     ```

2. Run the following command in Interactive SQL to create the RemoteOrders table.

   ```
   CREATE TABLE RemoteOrders
   (
    order_id           integer not null,
    product_id         integer not null,
   ```

```
 quantity           integer,
 order_status       varchar(10) default 'new',
 last_modified      timestamp default current timestamp,
 primary key(order_id)
)
```

Interactive SQL creates the RemoteOrders table in your consolidated database.

Stay connected in Interactive SQL for the following procedure.

Run MobiLink setup scripts

You can find setup scripts for each supported consolidated database in the *MobiLink/setup* subdirectory of your SQL Anywhere 11 installation.

In this procedure you set up a SQL Anywhere consolidated database. You do this using the *syncsa.sql* setup script. Running *syncsa.sql* creates a series of system tables and stored procedures prefaced with **ml_**. The MobiLink server works with these tables and stored procedures in the synchronization process.

To install MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. Run the following command in Interactive SQL to create MobiLink system tables and stored procedures. Replace *c:\Program Files\SQL Anywhere 11* with the location of your SQL Anywhere 11 installation.

   ```
   read "c:\Program Files\SQL Anywhere 11\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL applies *syncsa.sql* to your consolidated database.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about creating SQL Anywhere databases, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about creating tables, see "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*].

For information about setting up MobiLink consolidated databases, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

Lesson 3: Add synchronization scripts

In this lesson you add scripts to your consolidated database for SQL row handling and direct row handling.

SQL row handling

SQL row handling allows you to synchronize remote data with tables in your MobiLink consolidated database. SQL-based scripts define:

- How data that is uploaded from a MobiLink client is to be applied to the consolidated database.
- What data should be downloaded from the consolidated database.

In this lesson, you write synchronization scripts for the following SQL-based upload and download events.

- **upload_insert** To define how new orders inserted in a remote client database should be applied to the consolidated database.
- **download_cursor** To define what orders updated in the MobiLink consolidated database should be downloaded to remote clients.

In this procedure, you add synchronization script information to your MobiLink consolidated database using stored procedures.

To add SQL-based scripts to MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. Use the ml_add_table_script stored procedure to add SQL-based table scripts for the upload_insert and download_cursor events.

   Run the following commands in Interactive SQL. The upload_insert script inserts the uploaded order_id, product_id, quantity, and order_status into the MobiLink consolidated database. The download_cursor script uses timestamp-based filtering to download updated rows to remote clients.

   ```
   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'upload_insert',
        'INSERT INTO RemoteOrders( order_id, product_id, quantity,
   order_status)
         VALUES( ?, ?, ?, ? )' );

   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'download_cursor',
        'SELECT order_id, product_id, quantity, order_status
         FROM RemoteOrders WHERE last_modified >= ?');

   commit
   ```

Direct row handling processing

In this tutorial, you use direct row handling to add special processing to a SQL-based synchronization system. In this procedure you register method names corresponding to the handle_UploadData, handle_DownloadData, and end_download events. You create your own Java class in “Lesson 4: Creating a Java class using MobiLink direct row handling” on page 222.

To add information for direct row handling in MobiLink system tables

1. Connect to your consolidated database in Interactive SQL.

   Run the following command:

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. Register Java methods for the handle_UploadData and handle_DownloadData synchronization events.

   Execute the following commands in Interactive SQL.

   ```
   CALL ml_add_java_connection_script( 'default',
      'handle_UploadData',
        'MobiLinkOrders.GetUpload' );

   CALL ml_add_java_connection_script( 'default',
      'handle_DownloadData',
        'MobiLinkOrders.SetDownload' );

   commit
   ```

   Interactive SQL registers the user-defined GetUpload and SetDownload methods for the handle_UploadData and handle_DownloadData events, respectively.

Further reading

For information about using SQL-based events to upload data from remote clients to a MobiLink consolidated database, see:

- "Writing scripts to upload rows" [*MobiLink - Server Administration*]
- "upload_insert table event" [*MobiLink - Server Administration*]
- "upload_update table event" [*MobiLink - Server Administration*]
- "upload_delete table event" [*MobiLink - Server Administration*]

For information about uploading data to data sources other than consolidated databases, see "Handling direct uploads" [*MobiLink - Server Administration*].

For information about using SQL-based events to download data from a MobiLink consolidated database, see:

- "Writing scripts to download rows" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]
- "download_delete_cursor table event" [*MobiLink - Server Administration*]

For information about downloading data to data sources other than consolidated databases, see "Handling direct downloads" [*MobiLink - Server Administration*].

For information about the synchronization event sequence, see "Overview of MobiLink events" [*MobiLink - Server Administration*].

For information about synchronization techniques for download filtering, see "Timestamp-based downloads" [*MobiLink - Server Administration*] and "Partitioning rows among remote databases" [*MobiLink - Server Administration*].

For information about managing scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

For information about direct row handling, see “Direct row handling” [*MobiLink - Server Administration*].

Lesson 4: Creating a Java class using MobiLink direct row handling

In this lesson, you use direct row handling to process rows in the OrderComments table in your client database. You add the following methods for direct row handling:

- **GetUpload** You use this method for the handle_UploadData event. GetUpload writes uploaded comments to the XML file.

The following procedure shows you how to create a Java class including your methods for processing. For a complete listing, see "Complete MobiLinkOrders code listing (Java)" on page 226.

To create a Java class for download-only direct row handling

1. Create a class called MobilinkOrders using Java.

   Type the following code in a text editor or development environment:

   ```
   //Mobilink imports
   import ianywhere.ml.script.*;
   import java.io.*;
   import java.sql.*;

   //XML parser
   import javax.xml.parsers.DocumentBuilder;
   import javax.xml.parsers.DocumentBuilderFactory;
   import javax.xml.parsers.ParserConfigurationException;

   import org.xml.sax.SAXException;

   //DOM Objects
   import org.w3c.dom.Document;
   import org.w3c.dom.Element;
   import org.w3c.dom.Node;
   import org.w3c.dom.NodeList;

   //For writing XML objects
   import javax.xml.transform.Transformer;
   import javax.xml.transform.TransformerException;
   import javax.xml.transform.TransformerFactory;
   import javax.xml.transform.TransformerConfigurationException;
   import javax.xml.transform.dom.DOMSource;
   import javax.xml.transform.stream.StreamResult;

   public class MobiLinkOrders {
       // ...
   }
   ```

2. Declare a class-level DBConnectionContext instance and Document instance. Document is a class that represents an XML document as an object.

   ```
   DBConnectionContext _cc;
    Document _doc;
   ```

   The MobiLink server passes a DBConnectionContext instance to your class constructor. DBConnectionContext encapsulates information about the current connection with the MobiLink consolidated database.

3. Create your class constructor.

   Your class constructor sets your class-level DBConnectionContext instance.

   Type the following code in a text editor or development environment:

   ```
   public MobiLinkOrders( DBConnectionContext cc ) {
     _cc = cc;
   }
   ```

4. Write the GetUpload() method.

   a. Write the method declaration.

      Type the following code in a text editor or development environment:

      ```
      // method for the handle_UploadData synchronization event
      public void GetUpload( UploadData ut ) throws SQLException, IOException
      {
      ```

   b. Get any uploaded insert from the Mobilink client.

      Type the following code in a text editor or development environment:

      ```
      //  get an UploadedTableData for OrderComments
        UploadedTableData orderCommentsTbl =
          ut.getUploadedTableByName("OrderComments");

        // get inserts uploaded by the MobiLink client
        ResultSet insertResultSet = orderCommentsTbl.getInserts();
      ```

   c. Read in the existing XML file, **order_comments.xml**.

      Type the following code in a text editor or development environment:

      ```
            readDom("order_comments.xml");
      ```

   d. For each inserted row, add it to the XML file.

      Type the following code in a text editor or development environment:

      ```
            // Write out each insert in the XML file
            while( insertResultSet.next() ) {
              buildXML(insertResultSet);
            }
      ```

   e. Write the XML file.

      Type the following code in a text editor or development environment:

      ```
        writeXML();
      ```

   f. Close the ResultSet.

      Type the following code in a text editor or development environment:

      ```
            // Close the result set of uploaded inserts
            insertResultSet.close();
      ```

5. Write the buildXML method.

   Type the following code in a text editor or development environment:

   ```
   private void buildXML( ResultSet rs ) throws SQLException {
     int order_id = rs.getInt(1);
     int comment_id = rs.getInt(2);
   ```

```
        String order_comment = rs.getString(3);

        //Create the comment object to be added to the XML file
        Element comment = _doc.createElement("comment");
        comment.setAttribute("id", Integer.toString(comment_id));
        comment.appendChild(_doc.createTextNode(order_comment));

        //Get the root element (orders)
        Element root = _doc.getDocumentElement();

        //get each individual order
        NodeList rootChildren = root.getChildNodes();
        for(int i = 0; i < rootChildren.getLength(); i++) {
          //if the order exists, add the comment to the order
          Node n = rootChildren.item(i);
          if(n.getNodeType() == Node.ELEMENT_NODE) {
            Element e = (Element) n;
            int idIntVal = Integer.parseInt(e.getAttribute("id"));
            if(idIntVal == order_id) {
              e.appendChild(comment);
              //The comment has been added to the file, so exit the function
              return;
            }
          }
        }

        //if the order did not exist already, create it
        Element order = _doc.createElement("order");
        order.setAttribute("id", Integer.toString(order_id));
        //add the comment to the new order
        order.appendChild(comment);
        root.appendChild(order);
      }
```

6. Write the writeXML method.

   Type the following code in a text editor or development environment:

```
    private void writeXML() {

      try {

        // Use a Transformer for output
        TransformerFactory tFactory = TransformerFactory.newInstance();
        Transformer transformer = tFactory.newTransformer();

        //The XML source is _doc
        DOMSource source = new DOMSource(_doc);
        //write the xml data to order_comments.xml
        StreamResult result = new StreamResult(new
  File("order_comments.xml"));
        transformer.transform(source, result);

      } catch (TransformerConfigurationException tce) {
        // Error generated by the parser
        System.out.println ("\n** Transformer Factory error");
        System.out.println("   " + tce.getMessage() );

        // Use the contained exception, if any
        Throwable x = tce;
        if (tce.getException() != null) x = tce.getException();
        x.printStackTrace();

      } catch (TransformerException te) {
```



```
            // Error generated by the parser
            System.out.println ("\n** Transformation error");
            System.out.println("   " + te.getMessage() );

            // Use the contained exception, if any
            Throwable x = te;
            if (te.getException() != null) x = te.getException();
            x.printStackTrace();

        }

         }
```

7. Write the readDOM method.

   Type the following code in a text editor or development environment:

```
    private void readDom(String filename) {
      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        try {
          //parse the Document data into _doc
          DocumentBuilder builder = factory.newDocumentBuilder();
          _doc = builder.parse( new File(filename) );

        } catch (SAXException sxe) {
          // Error generated during parsing)
          Exception x = sxe;
          if (sxe.getException() != null) x = sxe.getException();
          x.printStackTrace();

        } catch (ParserConfigurationException pce) {
          // Parser with specified options can't be built
          pce.printStackTrace();

        } catch (IOException ioe) {
          // I/O error
          ioe.printStackTrace();
        }

    }
```

8. Save your Java code as *MobiLinkOrders.java* in your working directory *c:\MLobjxml*.
9. Compile your class file.
   - Navigate to the directory containing your Java source files.
   - Compile MobiLinkOrders with references to the MobiLink server API library for Java.

     For Java, you need to reference *mlscript.jar* in the *java* subdirectory of your SQL Anywhere installation. You also need to make sure that you have the XML DOM library installed correctly. Run the following command to compile your Java class, replacing *c:\Program Files\SQL Anywhere 11* with your SQL Anywhere 11 directory:

```
javac -classpath "c:\Program Files\SQL Anywhere 11\java\mlscript.jar"
MobiLinkOrders.java
```

Further reading

For more information about synchronization logic, see "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Complete MobiLinkOrders code listing (Java)

The following listing shows the complete Java MobiLinkOrders class listing used for this tutorial.

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

// For write operation
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

public class MobiLinkOrders {

  // class level DBConnectionContext
  DBConnectionContext _cc;
  Document _doc;

  public MobiLinkOrders( DBConnectionContext cc ) throws IOException,
FileNotFoundException {
    // declare a class-level DBConnectionContext
    _cc = cc;
  }


  //  method for the handle_UploadData synchronization event
  public void GetUpload( UploadData ut ) throws SQLException, IOException {
    // Get an UploadedTableData for the remote table
    UploadedTableData remoteOrdersTable =
ut.getUploadedTableByName("OrderComments");

    // Get inserts uploaded by the MobiLink client
    // as a java.sql.ResultSet
    ResultSet insertResultSet = remoteOrdersTable.getInserts();

    readDom("order_comments.xml");

    // Write out each insert in the XML file
    while( insertResultSet.next() ) {
      buildXML(insertResultSet);
    }
    writeXML();
```



```
    // Close the result set of uploaded inserts
    insertResultSet.close();

  }

  private void buildXML( ResultSet rs ) throws SQLException {
    int order_id = rs.getInt(1);
    int comment_id = rs.getInt(2);
    String order_comment = rs.getString(3);

    //Create the comment object to be added to the XML file
    Element comment = _doc.createElement("comment");
    comment.setAttribute("id", Integer.toString(comment_id));
    comment.appendChild(_doc.createTextNode(order_comment));

    //Get the root element (orders)
    Element root = _doc.getDocumentElement();

    //get each individual order
    NodeList rootChildren = root.getChildNodes();
    for(int i = 0; i < rootChildren.getLength(); i++) {
      //if the order exists, add the comment to the order
      Node n = rootChildren.item(i);
      if(n.getNodeType() == Node.ELEMENT_NODE) {
        Element e = (Element) n;
        int idIntVal = Integer.parseInt(e.getAttribute("id"));
        if(idIntVal == order_id) {
          e.appendChild(comment);
          //The comment has been added to the file, so exit the function
          return;
        }
      }
    }

    //if the order did not exist already, create it
    Element order = _doc.createElement("order");
    order.setAttribute("id", Integer.toString(order_id));
    //add the comment to the new order
    order.appendChild(comment);
    root.appendChild(order);
  }

  private void writeXML() {

    try {

      // Use a Transformer for output
      TransformerFactory tFactory = TransformerFactory.newInstance();
      Transformer transformer = tFactory.newTransformer();

      //The XML source is _doc
      DOMSource source = new DOMSource(_doc);
      //write the xml data to order_comments.xml
      StreamResult result = new StreamResult(new File("order_comments.xml"));
      transformer.transform(source, result);

    } catch (TransformerConfigurationException tce) {
      // Error generated by the parser
      System.out.println ("\n** Transformer Factory error");
      System.out.println("   " + tce.getMessage() );

      // Use the contained exception, if any
      Throwable x = tce;
```

```
            if (tce.getException() != null) x = tce.getException();
            x.printStackTrace();

          } catch (TransformerException te) {
            // Error generated by the parser
            System.out.println ("\n** Transformation error");
            System.out.println("   " + te.getMessage() );

            // Use the contained exception, if any
            Throwable x = te;
            if (te.getException() != null) x = te.getException();
            x.printStackTrace();

        }

         }

         private void readDom(String filename) {
           DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            try {
              //parse the Document data into _doc
              DocumentBuilder builder = factory.newDocumentBuilder();
              _doc = builder.parse( new File(filename) );

            } catch (SAXException sxe) {
              // Error generated during parsing)
              Exception x = sxe;
              if (sxe.getException() != null) x = sxe.getException();
              x.printStackTrace();

            } catch (ParserConfigurationException pce) {
              // Parser with specified options can't be built
              pce.printStackTrace();

            } catch (IOException ioe) {
              // I/O error
              ioe.printStackTrace();
            }

         }

       }
```


Lesson 5: Start the MobiLink server

In this lesson you start the MobiLink server. You start the MobiLink server (mlsrv11) using the -c option to connect to your consolidated database and the -sl java or -sl dnet option to load your Java class, respectively.

To start the MobiLink server for direct row handling

- Connect to your consolidated database and load or Java classes on the mlsrv11 command line.

  Run the following command. Replace *c:\MLobjxml* with the location of your Java source files.

  ```
  mlsrv11 -c "dsn=mlxml_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl java
  (-cp c:\MLobjxml)
  ```

  The MobiLink server window appears.

  Below is a description of each MobiLink server option used in this tutorial. The options -o, -v, and -dl provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v and -dl are typically not used in production.

Option	Description
-c	Precedes the connection string.
-o	Specifies the message log file *serverOut.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.
-dl	Displays all log messages on screen.
-zu+	Adds new users automatically.
-x	Sets the communications protocol and parameters for MobiLink clients.
-sl java	Specifies a set of directories to search for class files, and forces the Java Virtual Machine to load on server startup.
-sl dnet	Specifies the location of .NET assemblies and forces the CLR to load on server startup.

Further reading

For a complete list of MobiLink server options, see "MobiLink server options" [*MobiLink - Server Administration*].

For more information about loading Java classes see "-sl java option" [*MobiLink - Server Administration*].

Lesson 6: Set up your MobiLink client

In this tutorial, you use a SQL Anywhere database for your consolidated database and your MobiLink client. For tutorial purposes your MobiLink client, consolidated database, and MobiLink server all reside on the same computer.

To set up the MobiLink client database, you create the RemoteOrders and OrderComments tables. The RemoteOrders table corresponds to the RemoteOrders table on the consolidated database. The MobiLink server uses SQL-based scripts to synchronize remote orders. The OrderComments table is only used on client databases. The MobiLink server processes the OrderComments tables using special events.

You also create a synchronization user, publication, and subscription on the client database.

To set up your MobiLink client database

1. Create your MobiLink client database.

   In this lesson, you create a SQL Anywhere database using the dbinit command line utility.

   - To create your SQL Anywhere database, run the following command:

     ```
     dbinit -i -k remote1
     ```

     The -i and -k options tell dbinit to omit jConnect support and Watcom SQL compatibility views, respectively.

   - To start the database engine, run the following command:

     ```
     dbeng11 remote1
     ```

2. Connect to your MobiLink client database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

3. Create the RemoteOrders table.

   Execute the following command in Interactive SQL:

   ```
   create table RemoteOrders (
       order_id           integer not null,
       product_id         integer not null,
       quantity   integer,
       order_status       varchar(10) default 'new',
       primary key(order_id)
   )
   ```

4. Run the following command in Interactive SQL to create the OrderComments table:

   ```
   create table OrderComments (
       comment_id         integer not null,
       order_id           integer not null,
       order_comment      varchar (255),
       primary key(comment_id),
       foreign key (order_id) references
        RemoteOrders (order_id)
   )
   ```

5. Create your MobiLink synchronization user, publication, and subscription:


```
CREATE SYNCHRONIZATION USER ml_sales1;
CREATE PUBLICATION order_publ (TABLE RemoteOrders, Table OrderComments);
CREATE SYNCHRONIZATION SUBSCRIPTION TO order_publ FOR ml_sales1
 TYPE TCPIP ADDRESS 'host=localhost'
```

Note
You specify how to connect to the MobiLink server using the TYPE and ADDRESS clauses in the CREATE SYNCHRONIZATION SUBSCRIPTION statement.

You can use publications to determine what data is synchronized. In this case you specify the entire RemoteOrders and OrderComments tables.

Further reading

For information about creating a SQL Anywhere database, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about MobiLink clients, see "Introducing MobiLink clients" [*MobiLink - Client Administration*].

For information about creating MobiLink objects on the client, see:

- "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE PUBLICATION statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]

Lesson 7: Synchronize

The dbmlsync utility initiates MobiLink synchronization for SQL Anywhere remote databases. Before starting dbmlsync, add order data and comments to your remote database.

To set up your remote data (client-side)

1. Connect to the MobiLink client database in Interactive SQL:

   Run the following command:

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

2. Add an order to the RemoteOrders table in the client database.

   Execute the following command in Interactive SQL.

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. Add a comment to the OrderComments table in the client database.

   Execute the following command in Interactive SQL.

   ```
   INSERT INTO OrderComments (comment_id, order_id, order_comment)
    VALUES (1,1,'send promotional material with the order')
   ```

4. Commit your changes.

   Execute the following in Interactive SQL:

   ```
   COMMIT;
   ```

To start the synchronization client (client-side)

- At the command prompt, run the following command (on a single line):

  ```
  dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -e scn=on -o rem1.txt -v+
  ```

  Below is a description of each option:

Option	Description
-c	Specifies the connection string.
-e scn	Sets SendColumnNames to on. This is required by direct row handling if you want to reference columns by name.
-o	Specifies the message log file *rem1.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.

Once you have started the MobiLink synchronization client, an output screen appears indicating that the synchronization succeeded. SQL-based synchronization transferred rows in the client RemoteOrders table to the RemoteOrders table in the consolidated database.

Java processing inserted your comment in the XML file.

Further reading

For more information about dbmlsync, see "SQL Anywhere clients" [*MobiLink - Client Administration*].

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Close all instances of the following applications.
   - Interactive SQL
2. Close the SQL Anywhere, MobiLink, and synchronization client windows.
3. Delete all tutorial-related DSNs.
   - Start the ODBC Administrator.

     At a command prompt, type the following command:

     ```
     odbcad32
     ```
   - Remove the mlxml_db data source.
4. Delete the consolidated and remote databases.
   - Navigate to the directory containing your consolidated and remote databases.
   - Delete *MLconsolidated.db*, *MLconsolidated.log*, *remote1.db*, and *remote1.log*.

Further reading

For more information about MobiLink synchronization, see “Introducing MobiLink synchronization” on page 3.

For more information about MobiLink clients, see “Introducing MobiLink clients” [*MobiLink - Client Administration*].

For more information about MobiLink direct row handling, see “Direct row handling” [*MobiLink - Server Administration*].

Part III. Glossary

CHAPTER 14

Glossary

Adaptive Server Anywhere (ASA)

The relational database server component of SQL Anywhere Studio, intended for use in mobile and embedded environments or as a server for small and medium-sized businesses. In version 10.0.0, Adaptive Server Anywhere was renamed SQL Anywhere Server, and SQL Anywhere Studio was renamed SQL Anywhere.

See also: "SQL Anywhere" on page 262.

agent ID

See also: "client message store ID" on page 241.

article

In MobiLink or SQL Remote, an article is a database object that represents a whole table, or a subset of the columns and rows in a table. Articles are grouped together in a publication.

See also:

- "replication" on page 260
- "publication" on page 257

atomic transaction

A transaction that is guaranteed to complete successfully or not at all. If an error prevents part of an atomic transaction from completing, the transaction is rolled back to prevent the database from being left in an inconsistent state.

base table

Permanent tables for data. Tables are sometimes called **base tables** to distinguish them from temporary tables and views.

See also:

- "temporary table" on page 264
- "view" on page 266

bit array

A bit array is a type of array data structure that is used for efficient storage of a sequence of bits. A bit array is similar to a character string, except that the individual pieces are 0s (zeros) and 1s (ones) instead of characters. Bit arrays are typically used to hold a string of Boolean values.

business rule

A guideline based on real-world requirements. Business rules are typically implemented through check constraints, user-defined data types, and the appropriate use of transactions.

See also:

- "constraint" on page 242
- "user-defined data type" on page 266

carrier

A MobiLink object, stored in MobiLink system tables or a Notifier properties file, that contains information about a public carrier for use by server-initiated synchronization.

See also: "server-initiated synchronization" on page 262.

character set

A character set is a set of symbols, including letters, digits, spaces, and other symbols. An example of a character set is ISO-8859-1, also known as Latin1.

See also:

- "code page" on page 241
- "encoding" on page 246
- "collation" on page 241

check constraint

A restriction that enforces specified conditions on a column or set of columns.

See also:

- "constraint" on page 242
- "foreign key constraint" on page 248
- "primary key constraint" on page 257
- "unique constraint" on page 265

checkpoint

The point at which all changes to the database are saved to the database file. At other times, committed changes are saved only to the transaction log.

checksum

The calculated number of bits of a database page that is recorded with the database page itself. The checksum allows the database management system to validate the integrity of the page by ensuring that the numbers match as the page is being written to disk. If the counts match, it's assumed that page was successfully written.

client message store

In QAnywhere, a SQL Anywhere database on the remote device that stores messages.

client message store ID

In QAnywhere, a MobiLink remote ID that uniquely identifies a client message store.

client/server

A software architecture where one application (the client) obtains information from and sends information to another application (the server). The two applications often reside on different computers connected by a network.

code page

A code page is an encoding that maps characters of a character set to numeric representations, typically an integer between 0 and 255. An example of a code page is Windows code page 1252. For the purposes of this documentation, code page and encoding are interchangeable terms.

See also:

- “character set” on page 240
- “encoding” on page 246
- “collation” on page 241

collation

A combination of a character set and a sort order that defines the properties of text in the database. For SQL Anywhere databases, the default collation is determined by the operating system and language on which the server is running; for example, the default collation on English Windows systems is 1252LATIN1. A collation, also called a collating sequence, is used for comparing and sorting strings.

See also:

- “character set” on page 240
- “code page” on page 241
- “encoding” on page 246

command file

A text file containing SQL statements. Command files can be built manually, or they can be built automatically by database utilities. The dbunload utility, for example, creates a command file consisting of the SQL statements necessary to recreate a given database.

communication stream

In MobiLink, the network protocol used for communication between the MobiLink client and the MobiLink server.

concurrency

The simultaneous execution of two or more independent, and possibly competing, processes. SQL Anywhere automatically uses locking to isolate transactions and ensure that each concurrent application sees a consistent set of data.

See also:

- "transaction" on page 264
- "isolation level" on page 250

conflict resolution

In MobiLink, conflict resolution is logic that specifies what to do when two users modify the same row on different remote databases.

connection ID

A unique number that identifies a given connection between a client application and the database. You can determine the current connection ID using the following SQL statement:

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

connection-initiated synchronization

A form of MobiLink server-initiated synchronization in which synchronization is initiated when there are changes to connectivity.

See also: "server-initiated synchronization" on page 262.

connection profile

A set of parameters that are required to connect to a database, such as user name, password, and server name, that is stored and used as a convenience.

consolidated database

In distributed database environments, a database that stores the master copy of the data. In case of conflict or discrepancy, the consolidated database is considered to have the primary copy of the data.

See also:

- "synchronization" on page 264
- "replication" on page 260

constraint

A restriction on the values contained in a particular database object, such as a table or column. For example, a column may have a uniqueness constraint, which requires that all values in the column be different. A table may have a foreign key constraint, which specifies how the information in the table relates to data in some other table.

See also:

- "check constraint" on page 240
- "foreign key constraint" on page 248
- "primary key constraint" on page 257
- "unique constraint" on page 265

contention

The act of competing for resources. For example, in database terms, two or more users trying to edit the same row of a database contend for the rights to edit that row.

correlation name

The name of a table or view that is used in the FROM clause of a query—either its original name, or an alternate name, that is defined in the FROM clause.

creator ID

In UltraLite Palm OS applications, an ID that is assigned when the application is created.

cursor

A named linkage to a result set, used to access and update rows from a programming interface. In SQL Anywhere, cursors support forward and backward movement through the query results. Cursors consist of two parts: the cursor result set, typically defined by a SELECT statement; and the cursor position.

See also:

- “cursor result set” on page 243
- “cursor position” on page 243

cursor position

A pointer to one row within the cursor result set.

See also:

- “cursor” on page 243
- “cursor result set” on page 243

cursor result set

The set of rows resulting from a query that is associated with a cursor.

See also:

- “cursor” on page 243
- “cursor position” on page 243

data cube

A multi-dimensional result set with each dimension reflecting a different way to group and sort the same results. Data cubes provide complex information about data that would otherwise require self-join queries and correlated subqueries. Data cubes are a part of OLAP functionality.

data definition language (DDL)

The subset of SQL statements for defining the structure of data in the database. DDL statements create, modify, and remove database objects, such as tables and users.

data manipulation language (DML)

The subset of SQL statements for manipulating data in the database. DML statements retrieve, insert, update, and delete data in the database.

data type

The format of data, such as CHAR or NUMERIC. In the ANSI SQL standard, data types can also include a restriction on size, character set, and collation.

See also: "domain" on page 246.

database

A collection of tables that are related by primary and foreign keys. The tables hold the information in the database. The tables and keys together define the structure of the database. A database management system accesses this information.

See also:

- "foreign key" on page 247
- "primary key" on page 257
- "database management system (DBMS)" on page 244
- "relational database management system (RDBMS)" on page 259

database administrator (DBA)

The user with the permissions required to maintain the database. The DBA is generally responsible for all changes to a database schema, and for managing users and groups. The role of database administrator is automatically built into databases as user ID DBA with password sql.

database connection

A communication channel between a client application and the database. A valid user ID and password are required to establish a connection. The privileges granted to the user ID determine the actions that can be carried out during the connection.

database file

A database is held in one or more database files. There is an initial file, and subsequent files are called dbspaces. Each table, including its indexes, must be contained within a single database file.

See also: "dbspace" on page 245.

database management system (DBMS)

A collection of programs that allow you to create and use databases.

See also: "relational database management system (RDBMS)" on page 259.

database name

The name given to a database when it is loaded by a server. The default database name is the root of the initial database file.

See also: "database file" on page 244.

database object

A component of a database that contains or receives information. Tables, indexes, views, procedures, and triggers are database objects.

database owner (dbo)

A special user that owns the system objects not owned by SYS.

See also:

- "database administrator (DBA)" on page 244
- "SYS" on page 264

database server

A computer program that regulates all access to information in a database. SQL Anywhere provides two types of servers: network servers and personal servers.

DBA authority

The level of permission that enables a user to do administrative activity in the database. The DBA user has DBA authority by default.

See also: "database administrator (DBA)" on page 244.

dbspace

An additional database file that creates more space for data. A database can be held in up to 13 separate files (an initial file and 12 dbspaces). Each table, together with its indexes, must be contained in a single database file. The SQL command CREATE DBSPACE adds a new file to the database.

See also: "database file" on page 244.

deadlock

A state where a set of transactions arrives at a place where none can proceed.

device tracking

Functionality in MobiLink server-initiated synchronization that allows you to address messages using the MobiLink user name that identifies a remote device.

See also: "server-initiated synchronization" on page 262.

direct row handling

In MobiLink, a way to synchronize table data to sources other than the MobiLink-supported consolidated databases. You can implement both uploads and downloads with direct row handling.

See also:

- "consolidated database" on page 242
- "SQL-based synchronization" on page 262

domain

Aliases for built-in data types, including precision and scale values where applicable, and optionally including DEFAULT values and CHECK conditions. Some domains, such as the monetary data types, are pre-defined in SQL Anywhere. Also called user-defined data type.

See also: "data type" on page 244.

download

The stage in synchronization where data is transferred from the consolidated database to a remote database.

dynamic SQL

SQL that is generated programmatically by your program before it is executed. UltraLite dynamic SQL is a variant designed for small-footprint devices.

EBF

Express Bug Fix. An express bug fix is a subset of the software with one or more bug fixes. The bug fixes are listed in the release notes for the update. Bug fix updates may only be applied to installed software with the same version number. Some testing has been performed on the software, but the software has not undergone full testing. You should not distribute these files with your application unless you have verified the suitability of the software yourself.

embedded SQL

A programming interface for C programs. SQL Anywhere embedded SQL is an implementation of the ANSI and IBM standard.

encoding

Also known as character encoding, an encoding is a method by which each character in a character set is mapped onto one or more bytes of information, typically represented as a hexadecimal number. An example of an encoding is UTF-8.

See also:

- "character set" on page 240
- "code page" on page 241
- "collation" on page 241

event model

In MobiLink, the sequence of events that make up a synchronization, such as begin_synchronization and download_cursor. Events are invoked if a script is created for them.

external login

An alternate login name and password used when communicating with a remote server. By default, SQL Anywhere uses the names and passwords of its clients whenever it connects to a remote server on behalf of those clients. However, this default can be overridden by creating external logins. External logins are alternate login names and passwords used when communicating with a remote server.

extraction

In SQL Remote replication, the act of unloading the appropriate structure and data from the consolidated database. This information is used to initialize the remote database.

See also: "replication" on page 260.

failover

Switching to a redundant or standby server, system, or network on failure or unplanned termination of the active server, system, or network. Failover happens automatically.

FILE

In SQL Remote replication, a message system that uses shared files for exchanging replication messages. This is useful for testing and for installations without an explicit message-transport system.

See also:"replication" on page 260

file-based download

In MobiLink, a way to synchronize data in which downloads are distributed as files, allowing offline distribution of synchronization changes.

file-definition database

In MobiLink, a SQL Anywhere database that is used for creating download files.

See also: "file-based download" on page 247.

foreign key

One or more columns in a table that duplicate the primary key values in another table. Foreign keys establish relationships between tables.

See also:

- "primary key" on page 257
- "foreign table" on page 248

foreign key constraint

A restriction on a column or set of columns that specifies how the data in the table relates to the data in some other table. Imposing a foreign key constraint on a set of columns makes those columns the foreign key.

See also:

- "constraint" on page 242
- "check constraint" on page 240
- "primary key constraint" on page 257
- "unique constraint" on page 265

foreign table

The table containing the foreign key.

See also: "foreign key" on page 247.

full backup

A backup of the entire database, and optionally, the transaction log. A full backup contains all the information in the database and thus provides protection in the event of a system or media failure.

See also: "incremental backup" on page 249.

gateway

A MobiLink object, stored in MobiLink system tables or a Notifier properties file, that contains information about how to send messages for server-initiated synchronization.

See also: "server-initiated synchronization" on page 262.

generated join condition

A restriction on join results that is automatically generated. There are two types: key and natural. Key joins are generated when you specify KEY JOIN or when you specify the keyword JOIN but do not use the keywords CROSS, NATURAL, or ON. For a key join, the generated join condition is based on foreign key relationships between tables. Natural joins are generated when you specify NATURAL JOIN; the generated join condition is based on common column names in the two tables.

See also:

- "join" on page 251
- "join condition" on page 251

generation number

In MobiLink, a mechanism for forcing remote databases to upload data before applying any more download files.

See also: "file-based download" on page 247.

global temporary table

A type of temporary table for which data definitions are visible to all users until explicitly dropped. Global temporary tables let each user open their own identical instance of a table. By default, rows are deleted on commit, and rows are always deleted when the connection is ended.

See also:

- "temporary table" on page 264
- "local temporary table" on page 252

grant option

The level of permission that allows a user to grant permissions to other users.

hash

A hash is an index optimization that transforms index entries into keys. An index hash aims to avoid the expensive operation of finding, loading, and then unpacking the rows to determine the indexed value, by including enough of the actual row data with its row ID.

histogram

The most important component of column statistics, histograms are a representation of data distribution. SQL Anywhere maintains histograms to provide the optimizer with statistical information about the distribution of values in columns.

iAnywhere JDBC driver

The iAnywhere JDBC driver provides a JDBC driver that has some performance benefits and feature benefits compared to the pure Java jConnect JDBC driver, but which is not a pure-Java solution. The iAnywhere JDBC driver is recommended in most cases.

See also:

- "JDBC" on page 251
- "jConnect" on page 251

identifier

A string of characters used to reference a database object, such as a table or column. An identifier may contain any character from A through Z, a through z, 0 through 9, underscore (_), at sign (@), number sign (#), or dollar sign ($).

incremental backup

A backup of the transaction log only, typically used between full backups.

See also: "transaction log" on page 265.

index

A sorted set of keys and pointers associated with one or more columns in a base table. An index on one or more columns of a table can improve performance.

InfoMaker

A reporting and data maintenance tool that lets you create sophisticated forms, reports, graphs, cross-tabs, and tables, as well as applications that use these reports as building blocks.

inner join

A join in which rows appear in the result set only if both tables satisfy the join condition. Inner joins are the default.

See also:

- "join" on page 251
- "outer join" on page 255

integrated login

A login feature that allows the same single user ID and password to be used for operating system logins, network logins, and database connections.

integrity

Adherence to rules that ensure that data is correct and accurate, and that the relational structure of the database is intact.

See also: "referential integrity" on page 259.

Interactive SQL

A SQL Anywhere application that allows you to query and alter data in your database, and modify the structure of your database. Interactive SQL provides a pane for you to enter SQL statements, as well as panes that return information about how the query was processed and the result set.

isolation level

The degree to which operations in one transaction are visible to operations in other concurrent transactions. There are four isolation levels, numbered 0 through 3. Level 3 provides the highest level of isolation. Level 0 is the default setting. SQL Anywhere also supports three snapshot isolation levels: snapshot, statement-snapshot, and readonly-statement-snapshot.

See also: "snapshot isolation" on page 262.

JAR file

Java archive file. A compressed file format consisting of a collection of one or more packages used for Java applications. It includes all the resources necessary to install and run a Java program in a single compressed file.

Java class

The main structural unit of code in Java. It is a collection of procedures and variables grouped together because they all relate to a specific, identifiable category.

jConnect

A Java implementation of the JavaSoft JDBC standard. It provides Java developers with native database access in multi-tier and heterogeneous environments. However, the iAnywhere JDBC driver is the preferred JDBC driver for most cases.

See also:

- "JDBC" on page 251
- "iAnywhere JDBC driver" on page 249

JDBC

Java Database Connectivity. A SQL-language programming interface that allows Java applications to access relational data. The preferred JDBC driver is the iAnywhere JDBC driver.

See also:

- "jConnect" on page 251
- "iAnywhere JDBC driver" on page 249

join

A basic operation in a relational system that links the rows in two or more tables by comparing the values in specified columns.

join condition

A restriction that affects join results. You specify a join condition by inserting an ON clause or WHERE clause immediately after the join. In the case of natural and key joins, SQL Anywhere generates a join condition.

See also:

- "join" on page 251
- "generated join condition" on page 248

join type

SQL Anywhere provides four types of joins: cross join, key join, natural join, and joins using an ON clause.

See also: "join" on page 251.

Listener

A program, dblsn, that is used by MobiLink server-initiated synchronization. Listeners are installed on remote devices and configured to initiate actions on the device when they receive information from a Notifier.

See also: “server-initiated synchronization” on page 262.

local temporary table

A type of temporary table that exists only for the duration of a compound statement or until the end of the connection. Local temporary tables are useful when you need to load a set of data only once. By default, rows are deleted on commit.

See also:

- “temporary table” on page 264
- “global temporary table” on page 249

lock

A concurrency control mechanism that protects the integrity of data during the simultaneous execution of multiple transactions. SQL Anywhere automatically applies locks to prevent two connections from changing the same data at the same time, and to prevent other connections from reading data that is in the process of being changed.

You control locking by setting the isolation level.

See also:

- “isolation level” on page 250
- “concurrency” on page 241
- “integrity” on page 250

log file

A log of transactions maintained by SQL Anywhere. The log file is used to ensure that the database is recoverable in the event of a system or media failure, to improve database performance, and to allow data replication using SQL Remote.

See also:

- “transaction log” on page 265
- “transaction log mirror” on page 265
- “full backup” on page 248

logical index

A reference (pointer) to a physical index. There is no indexing structure stored on disk for a logical index.

LTM

Log Transfer Manager (LTM) also called Replication Agent. Used with Replication Server, the LTM is the program that reads a database transaction log and sends committed changes to Sybase Replication Server.

See: “Replication Server” on page 260.

maintenance release

A maintenance release is a complete set of software that upgrades installed software from an older version with the same major version number (version number format is *major.minor.patch.build*). Bug fixes and other changes are listed in the release notes for the upgrade.

materialized view

A materialized view is a view that has been computed and stored on disk. Materialized views have characteristics of both views (they are defined using a query specification), and of tables (they allow most table operations to be performed on them).

See also:

- "base table" on page 239
- "view" on page 266

message log

A log where messages from an application such as a database server or MobiLink server can be stored. This information can also appear in a messages window or be logged to a file. The message log includes informational messages, errors, warnings, and messages from the MESSAGE statement.

message store

In QAnywhere, databases on the client and server device that store messages.

See also:

- "client message store" on page 240
- "server message store" on page 262

message system

In SQL Remote replication, a protocol for exchanging messages between the consolidated database and a remote database. SQL Anywhere includes support for the following message systems: FILE, FTP, and SMTP.

See also:

- "replication" on page 260
- "FILE" on page 247

message type

In SQL Remote replication, a database object that specifies how remote users communicate with the publisher of a consolidated database. A consolidated database may have several message types defined for it; this allows different remote users to communicate with it using different message systems.

See also:

- "replication" on page 260
- "consolidated database" on page 242

metadata

Data about data. Metadata describes the nature and content of other data.

See also: "schema" on page 261.

mirror log

See also: "transaction log mirror" on page 265.

MobiLink

A session-based synchronization technology designed to synchronize UltraLite and SQL Anywhere remote databases with a consolidated database.

See also:

- "consolidated database" on page 242
- "synchronization" on page 264
- "UltraLite" on page 265

MobiLink client

There are two kinds of MobiLink clients. For SQL Anywhere remote databases, the MobiLink client is the dbmlsync command line utility. For UltraLite remote databases, the MobiLink client is built in to the UltraLite runtime library.

MobiLink Monitor

A graphical tool for monitoring MobiLink synchronizations.

MobiLink server

The computer program that runs MobiLink synchronization, mlsrv11.

MobiLink system table

System tables that are required by MobiLink synchronization. They are installed by MobiLink setup scripts into the MobiLink consolidated database.

MobiLink user

A MobiLink user is used to connect to the MobiLink server. You create the MobiLink user on the remote database and register it in the consolidated database. MobiLink user names are entirely independent of database user names.

network protocol

The type of communication, such as TCP/IP or HTTP.

network server

A database server that accepts connections from computers sharing a common network.

See also: "personal server" on page 256.

normalization

The refinement of a database structure to eliminate redundancy and improve organization according to rules based on relational database theory.

Notifier

A program that is used by MobiLink server-initiated synchronization. Notifiers run on the same computer as the MobiLink server. They poll the consolidated database for push requests, and then send notifications to Listeners.

See also:

- "server-initiated synchronization" on page 262
- "Listener" on page 251

object tree

In Sybase Central, the hierarchy of database objects. The top level of the object tree shows all products that your version of Sybase Central supports. Each product expands to reveal its own sub-tree of objects.

See also: "Sybase Central" on page 264.

ODBC

Open Database Connectivity. A standard Windows interface to database management systems. ODBC is one of several interfaces supported by SQL Anywhere.

ODBC Administrator

A Microsoft program included with Windows operating systems for setting up ODBC data sources.

ODBC data source

A specification of the data a user wants to access via ODBC, and the information needed to get to that data.

outer join

A join that preserves all the rows in a table. SQL Anywhere supports left, right, and full outer joins. A left outer join preserves the rows in the table to the left of the join operator, and returns a null when a row in the right table does not satisfy the join condition. A full outer join preserves all the rows from both tables.

See also:

- "join" on page 251
- "inner join" on page 250

package

In Java, a collection of related classes.

parse tree

An algebraic representation of a query.

PDB

A Palm database file.

performance statistic

A value reflecting the performance of the database system. The CURRREAD statistic, for example, represents the number of file reads issued by the engine that have not yet completed.

personal server

A database server that runs on the same computer as the client application. A personal database server is typically used by a single user on a single computer, but it can support several concurrent connections from that user.

physical index

The actual indexing structure of an index, as it is stored on disk.

plug-in module

In Sybase Central, a way to access and administer a product. Plug-ins are usually installed and registered automatically with Sybase Central when you install the respective product. Typically, a plug-in appears as a top-level container, in the Sybase Central main window, using the name of the product itself; for example, SQL Anywhere.

See also: "Sybase Central" on page 264.

policy

In QAnywhere, the way you specify when message transmission should occur.

polling

In MobiLink server-initiated synchronization, the way the Notifier detects push requests on the consolidated database.

See also: "server-initiated synchronization" on page 262.

PowerDesigner

A database modeling application. PowerDesigner provides a structured approach to designing a database or data warehouse. SQL Anywhere includes the Physical Data Model component of PowerDesigner.

PowerJ

A Sybase product for developing Java applications.

predicate

A conditional expression that is optionally combined with the logical operators AND and OR to make up the set of conditions in a WHERE or HAVING clause. In SQL, a predicate that evaluates to UNKNOWN is interpreted as FALSE.

primary key

A column or list of columns whose values uniquely identify every row in the table.

See also: “foreign key” on page 247.

primary key constraint

A uniqueness constraint on the primary key columns. A table can have only one primary key constraint.

See also:

- “constraint” on page 242
- “check constraint” on page 240
- “foreign key constraint” on page 248
- “unique constraint” on page 265
- “integrity” on page 250

primary table

The table containing the primary key in a foreign key relationship.

proxy table

A local table containing metadata used to access a table on a remote database server as if it were a local table.

See also: “metadata” on page 254.

publication

In MobiLink or SQL Remote, a database object that identifies data that is to be synchronized. In MobiLink, publications exist only on the clients. A publication consists of articles. SQL Remote users can receive a publication by subscribing to it. MobiLink users can synchronize a publication by creating a synchronization subscription to it.

See also:

- “replication” on page 260
- “article” on page 239
- “publication update” on page 258

publication update

In SQL Remote replication, a list of changes made to one or more publications in one database. A publication update is sent periodically as part of a replication message to the remote database(s).

See also:

- "replication" on page 260
- "publication" on page 257

publisher

In SQL Remote replication, the single user in a database who can exchange replication messages with other replicating databases.

See also: "replication" on page 260.

push notification

In QAnywhere, a special message delivered from the server to a QAnywhere client that prompts the client to initiate a message transmission.

See also: "QAnywhere" on page 258.

push request

In MobiLink server-initiated synchronization, a row in a SQL result set or table on the consolidated database that contains a notification and information about how to send the notification.

See also: "server-initiated synchronization" on page 262.

QAnywhere

Application-to-application messaging, including mobile device to mobile device and mobile device to and from the enterprise, that permits communication between custom programs running on mobile or wireless devices and a centrally located server application.

QAnywhere agent

In QAnywhere, a process running on the client device that monitors the client message store and determines when message transmission should occur.

query

A SQL statement or group of SQL statements that access and/or manipulate data in a database.

See also: "SQL" on page 262.

Redirector

A web server plug-in that routes requests and responses between a client and the MobiLink server. This plug-in also implements load-balancing and failover mechanisms.

reference database

In MobiLink, a SQL Anywhere database used in the development of UltraLite clients. You can use a single SQL Anywhere database as both reference and consolidated database during development. Databases made with other products cannot be used as reference databases.

referencing object

An object, such as a view, whose definition directly references another object in the database, such as a table.

See also: "foreign key" on page 247.

referenced object

An object, such as a table, that is directly referenced in the definition of another object, such as a view.

See also: "primary key" on page 257.

referential integrity

Adherence to rules governing data consistency, specifically the relationships between the primary and foreign key values in different tables. To have referential integrity, the values in each foreign key must correspond to the primary key values of a row in the referenced table.

See also:

- "primary key" on page 257
- "foreign key" on page 247

regular expression

A regular expression is a sequence of characters, wildcards, and operators that defines a pattern to search for within a string.

relational database management system (RDBMS)

A type of database management system that stores data in the form of related tables.

See also: "database management system (DBMS)" on page 244.

remote database

In MobiLink or SQL Remote, a database that exchanges data with a consolidated database. Remote databases may share all or some of the data in the consolidated database.

See also:

- "synchronization" on page 264
- "consolidated database" on page 242

remote DBA authority

In SQL Remote, a level of permission required by the Message Agent. In MobiLink, a level of permission required by the SQL Anywhere synchronization client (dbmlsync). When the Message Agent or synchronization client connects as a user who has this authority, it has full DBA access. The user ID has no additional permissions when not connected through the Message Agent or synchronization client.

See also: “DBA authority” on page 245.

remote ID

A unique identifier in SQL Anywhere and UltraLite databases that is used by MobiLink. The remote ID is initially set to NULL and is set to a GUID during a database's first synchronization.

replication

The sharing of data among physically distinct databases. Sybase has three replication technologies: MobiLink, SQL Remote, and Replication Server.

Replication Agent

See: “LTM” on page 252.

replication frequency

In SQL Remote replication, a setting for each remote user that determines how often the publisher's message agent should send replication messages to that remote user.

See also: “replication” on page 260.

replication message

In SQL Remote or Replication Server, a communication sent between a publishing database and a subscribing database. Messages contain data, passthrough statements, and information required by the replication system.

See also:

- “replication” on page 260
- “publication update” on page 258

Replication Server

A Sybase connection-based replication technology that works with SQL Anywhere and Adaptive Server Enterprise. It is intended for near-real time replication between a small number of databases.

See also: “LTM” on page 252.

role

In conceptual database modeling, a verb or phrase that describes a relationship from one point of view. You can describe each relationship with two roles. Examples of roles are "contains" and "is a member of."

role name

The name of a foreign key. This is called a role name because it names the relationship between the foreign table and primary table. By default, the role name is the table name, unless another foreign key is already using that name, in which case the default role name is the table name followed by a three-digit unique number. You can also create the role name yourself.

See also: "foreign key" on page 247.

rollback log

A record of the changes made during each uncommitted transaction. In the event of a ROLLBACK request or a system failure, uncommitted transactions are reversed out of the database, returning the database to its former state. Each transaction has a separate rollback log, which is deleted when the transaction is complete.

See also: "transaction" on page 264.

row-level trigger

A trigger that executes once for each row that is changed.

See also:

- "trigger" on page 265
- "statement-level trigger" on page 263

schema

The structure of a database, including tables, columns, and indexes, and the relationships between them.

script

In MobiLink, code written to handle MobiLink events. Scripts programmatically control data exchange to meet business needs.

See also: "event model" on page 247.

script-based upload

In MobiLink, a way to customize the upload process as an alternative to using the log file.

script version

In MobiLink, a set of synchronization scripts that are applied together to create a synchronization.

secured feature

A feature specified by the -sf option when a database server is started, so it is not available for any database running on that database server.

server-initiated synchronization

A way to initiate MobiLink synchronization programmatically from the consolidated database.

server management request

A QAnywhere message that is formatted as XML and sent to the QAnywhere system queue as a way to administer the sever message store or monitor QAnywhere applications.

server message store

In QAnywhere, a relational database on the server that temporarily stores messages until they are transmitted to a client message store or JMS system. Messages are exchanged between clients via the server message store.

service

In Windows operating systems, a way of running applications when the user ID running the application is not logged on.

session-based synchronization

A type of synchronization where synchronization results in consistent data representation across both the consolidated and remote databases. MobiLink is session-based.

snapshot isolation

A type of isolation level that returns a committed version of the data for transactions that issue read requests. SQL Anywhere provides three snapshot isolation levels: snapshot, statement-snapshot, and readonly-statement-snapshot. When using snapshot isolation, read operations do not block write operations.

See also: "isolation level" on page 250.

SQL

The language used to communicate with relational databases. ANSI has defined standards for SQL, the latest of which is SQL-2003. SQL stands, unofficially, for Structured Query Language.

SQL Anywhere

The relational database server component of SQL Anywhere that is intended for use in mobile and embedded environments or as a server for small and medium-sized businesses. SQL Anywhere is also the name of the package that contains the SQL Anywhere RDBMS, the UltraLite RDBMS, MobiLink synchronization software, and other components.

SQL-based synchronization

In MobiLink, a way to synchronize table data to MobiLink-supported consolidated databases using MobiLink events. For SQL-based synchronization, you can use SQL directly or you can return SQL using the MobiLink server APIs for Java and .NET.

SQL Remote

A message-based data replication technology for two-way replication between consolidated and remote databases. The consolidated and remote databases must be SQL Anywhere.

SQL statement

A string containing SQL keywords designed for passing instructions to a DBMS.

See also:

- "schema" on page 261
- "SQL" on page 262
- "database management system (DBMS)" on page 244

statement-level trigger

A trigger that executes after the entire triggering statement is completed.

See also:

- "trigger" on page 265
- "row-level trigger" on page 261

stored procedure

A program comprised of a sequence of SQL instructions, stored in the database and used to perform a particular task.

string literal

A string literal is a sequence of characters enclosed in single quotes.

subquery

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

There are two types of subquery: correlated and nested.

subscription

In MobiLink synchronization, a link in a client database between a publication and a MobiLink user, allowing the data described by the publication to be synchronized.

In SQL Remote replication, a link between a publication and a remote user, allowing the user to exchange updates on that publication with the consolidated database.

See also:

- "publication" on page 257
- "MobiLink user" on page 254

Sybase Central

A database management tool that provides SQL Anywhere database settings, properties, and utilities in a graphical user interface. Sybase Central can also be used for managing other Sybase products, including MobiLink.

synchronization

The process of replicating data between databases using MobiLink technology.

In SQL Remote, synchronization is used exclusively to denote the process of initializing a remote database with an initial set of data.

See also:

- "MobiLink" on page 254
- "SQL Remote" on page 263

SYS

A special user that owns most of the system objects. You cannot log in as SYS.

system object

Database objects owned by SYS or dbo.

system table

A table, owned by SYS or dbo, that holds metadata. System tables, also known as data dictionary tables, are created and maintained by the database server.

system view

A type of view, included in every database, that presents the information held in the system tables in an easily understood format.

temporary table

A table that is created for the temporary storage of data. There are two types: global and local.

See also:

- "local temporary table" on page 252
- "global temporary table" on page 249

transaction

A sequence of SQL statements that comprise a logical unit of work. A transaction is processed in its entirety or not at all. SQL Anywhere supports transaction processing, with locking features built in to allow concurrent transactions to access the database without corrupting the data. Transactions end either with a COMMIT statement, which makes the changes to the data permanent, or a ROLLBACK statement, which undoes all the changes made during the transaction.

transaction log

A file storing all changes made to a database, in the order in which they are made. It improves performance and allows data recovery in the event the database file is damaged.

transaction log mirror

An optional identical copy of the transaction log file, maintained simultaneously. Every time a database change is written to the transaction log file, it is also written to the transaction log mirror file.

A mirror file should be kept on a separate device from the transaction log, so that if either device fails, the other copy of the log keeps the data safe for recovery.

See also: "transaction log" on page 265.

transactional integrity

In MobiLink, the guaranteed maintenance of transactions across the synchronization system. Either a complete transaction is synchronized, or no part of the transaction is synchronized.

transmission rule

In QAnywhere, logic that determines when message transmission is to occur, which messages to transmit, and when messages should be deleted.

trigger

A special form of stored procedure that is executed automatically when a user runs a query that modifies the data.

See also:

- "row-level trigger" on page 261
- "statement-level trigger" on page 263
- "integrity" on page 250

UltraLite

A database optimized for small, mobile, and embedded devices. Intended platforms include cell phones, pagers, and personal organizers.

UltraLite runtime

An in-process relational database management system that includes a built-in MobiLink synchronization client. The UltraLite runtime is included in the libraries used by each of the UltraLite programming interfaces, as well as in the UltraLite engine.

unique constraint

A restriction on a column or set of columns requiring that all non-null values are different. A table can have multiple unique constraints.

See also:

- "foreign key constraint" on page 248
- "primary key constraint" on page 257
- "constraint" on page 242

unload

Unloading a database exports the structure and/or data of the database to text files (SQL command files for the structure, and ASCII comma-separated files for the data). You unload a database with the Unload utility.

In addition, you can unload selected portions of your data using the UNLOAD statement.

upload

The stage in synchronization where data is transferred from a remote database to a consolidated database.

user-defined data type

See "domain" on page 246.

validate

To test for particular types of file corruption of a database, table, or index.

view

A SELECT statement that is stored in the database as an object. It allows users to see a subset of rows or columns from one or more tables. Each time a user uses a view of a particular table, or combination of tables, it is recomputed from the information stored in those tables. Views are useful for security purposes, and to tailor the appearance of database information to make data access straightforward.

window

The group of rows over which an analytic function is performed. A window may contain one, many, or all rows of data that has been partitioned according to the grouping specifications provided in the window definition. The window moves to include the number or range of rows needed to perform the calculations for the current row in the input. The main benefit of the window construct is that it allows additional opportunities for grouping and analysis of results, without having to perform additional queries.

Windows

The Microsoft Windows family of operating systems, such as Windows Vista, Windows XP, and Windows 200x.

Windows CE

See "Windows Mobile" on page 266.

Windows Mobile

A family of operating systems produced by Microsoft for mobile devices.

work table

An internal storage area for interim results during query optimization.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

