iAnywhere®
A SYBASE COMPANY

Populating or Re-Populating a Consolidated Database Using MobiLink

*A whitepaper from iAnywhere Solutions, Inc.,
a subsidiary of Sybase, Inc.*

Contents

Overview

This document describes a technique that allows you to synchronize all the rows of remote Adaptive Server Anywhere databases to a consolidated database, even if the rows have been previously synchronized. You may want to do this to put data into a new consolidated database or to recover data into an existing consolidated database using data from the remote databases. The technique consists of unloading and reloading the remote Adaptive Server Anywhere databases in a way that marks the remote data as unsynchronized changes, and then synchronizing using a special set of synchronization scripts that either update or insert the remote data into the consolidated database.

When is populating or re-populating the consolidated database required?

There are a number of different scenarios where you might want to use the data in remote MobiLink client databases to populate or re-populate your MobiLink consolidated database. Following are some examples:

Case 1: You lost your consolidated database and do *not* have a backup.

- ♦ This situation, of course, should never occur. In a distributed environment (MobiLink or SQL Remote), it is essential to guard against failure of the consolidated database. Failure can occur in many ways, including hardware problems (drive crash), database corruption, or faulty backup and recovery procedures.

Case 2: You have a franchise and are adding one store at a time.

- ♦ Assume you are adding one database a week to a MobiLink setup. Each store has some data that does not necessarily exist on the consolidated database:
  - A list of the products that are common to every store
  - Orders that are unique to the store itself
- ♦ This example assumes that the primary keys of the records do not conflict with any other store's primary keys.

Case 3: Remotes own the data and the consolidated database is merely a reporting center. In this scenario,

- ♦ The consolidated database is being used for data collection and reporting purposes; all its data comes from the remotes.
- ♦ If the consolidated database is lost, it really does not matter since it does not control the data; it merely collects the data together. In this scenario, the consolidated database can be rebuilt at any time.

What version of the software can I use?

This document assumes that both MobiLink and your Adaptive Server Anywhere remote databases are version 8. UltraLite remotes cannot be used with this technique. You can also use this technique with version 7.0.3 and later of the software. When using version 7, the technique is the same, but the MobiLink SQL statements that you must use are different. For example, you must use CREATE SYNCHRONIZATION DEFINITION in version 7 and CREATE SYNCHRONIZATION SUBSCRIPTION/CREATE PUBLICATION in version 8.

Requirements

- ♦ In a normal MobiLink setup you have one consolidated database, which can be any of the following: Sybase Adaptive Server Anywhere, Sybase Adaptive Server Enterprise, Microsoft SQL Server, Oracle, or IBM DB2.
- ♦ You can have one or more remote Adaptive Server Anywhere databases.
- ♦ In MobiLink you must define at least one script version for your synchronization logic. The version is used to define a set of synchronization scripts. A MobiLink server can allow remotes running different script versions to all synchronize at the same time. The application itself can have multiple script versions. For example, you could have script version 1.0.0 or 2.0.0, or intermediate version 1.5.0. Alternatively, you could have script versions called Salesperson, Support, and Manager. In this example the remote uses two different script versions.
- ♦ You can follow these directions even if you already have a setup that uses MobiLink, as described in Case 1 in “When is populating or re-populating the consolidated database required?” on page 2.

The process outlined here is intended for one time usage. Once the consolidated database is rebuilt, your usual process should be used (that is, everything goes back to the way it was before the rebuild).

Step by step example

The basic technique is to unload and reload the remote database in such a way that the loading is recorded in the log file. This way, all the data is marked as inserts that are not yet synchronized. Once this is done, you synchronize, and the rows with new primary keys are inserted, while rows with duplicate primary keys are handled in a manner consistent with your business logic.

The following sections describe the steps in more detail. You should use a new script version to differentiate these synchronization scripts from your normal synchronization scripts.

On the consolidated database

The first thing you need to determine is whether you need to handle the case where rows already exist on the consolidated database. You need to handle this case unless you are certain that all the remotes have rows with different primary keys and no rows with those primary keys are already in the consolidated database. Otherwise, an upload from a remote may try to insert a row with a primary key that matches a row already in the consolidated database.

Note
If your synchronization logic is not the same as your regular synchronization logic, you should use a new script version name for it. The examples in the following sections use the version name REBUILD.

♦ If you are certain you do not have rows in remote databases that have the same primary keys as rows in other remotes or the consolidated database, then you can use your normal synchronization scripts. In this case, proceed to "On the remotes" on page 8.

♦ If you need to handle rows with matching primary keys and your consolidated database is Adaptive Server Anywhere 8.0.2 or higher, then you can you can use your normal synchronization scripts, provided that your upload_insert script uses an INSERT statement with the ON EXISTING clause to handle rows that already exist. In this case, proceed to "Using the ON EXISTING clause for INSERT" on page 4.

♦ Otherwise, you should use a new script version that utilizes forced-conflict synchronization to efficiently handle row conflicts. In this case, proceed to "Forced-conflict synchronizations" on page 5.

Using the ON EXISTING clause for INSERT

If your consolidated database is Adaptive Server Anywhere version 8.0.2 or higher, you can use the ON EXISTING clause in the INSERT statement of your upload_insert script. Specifying the ON EXISTING clause lets you choose one of the following actions for rows that already exist in the table:

♦ Generate an error for duplicate key values. This is the default behavior if the ON EXISTING clause is not specified.

♦ Silently ignore the input row, without generating any errors. Use ON EXISTING SKIP to specify this behavior.

♦ Update the existing row with the values in the input row. Use ON EXISTING UPDATE to specify this behavior.

If none of these actions fit your requirements, then you should use forced-conflict synchronizations instead of the ON EXISTING clause.

☞ For information about forced-conflict synchronizations, see "Forced-conflict synchronizations" on page 5.

ON EXISTING clause example

Assume that the following table is being synchronized from the remote database:

```
CREATE TABLE Product (
    prod_id          INTEGER NOT NULL PRIMARY KEY,
    price            INTEGER,
    prod_name        VARCHAR(30),
    )
```

The Product table has only one event defined:

upload_insert (Script Version = REBUILD)

```
INSERT INTO Product( prod_id, price, prod_name )
ON EXISTING UPDATE VALUES (?, ?, ?)
```

The above INSERT statement inserts new rows that do not already exist on the consolidated database and updates any rows that do exist on the consolidated database with the same primary key.

Forced-conflict synchronizations

The MobiLink conflict resolution capabilities give you great flexibility in handling how data that is uploaded from remote databases is applied to the consolidated database. Before describing how to use forced-conflict synchronizations, it is important for you to understand MobiLink conflicts.

- What is a conflict?

  A conflict occurs if a remote database uploads a record that has changed on the consolidated database since the last time that remote synchronized. By default, MobiLink resolves the conflict by accepting the uploaded record. You have the ability to override MobiLinkâ€™s default conflict resolution and programmatically choose which values to use from the remote. Conflict resolution only affects rows that were uploaded as UPDATEs. Inserts and deletes are not included in MobiLinkâ€™s conflict resolution.

- What is a forced conflict?

  In order to accomplish a rebuild of the consolidated database, you must use a feature of MobiLink called forced conflicts. Forced conflicts allow you to tell MobiLink what to do with each and every row uploaded from the remote database. Unlike normal conflict resolution, forced conflicts can be used for inserts, updates, and deletes. The MobiLink scripts you write at the consolidated database can determine whether to perform an insert or an update for each table, or even base an operation on the data on a row by row basis.

  When using forced conflicts, all rows coming from the remote database are inserted into the consolidated database. The rows are inserted even if the operation at the remote was an update or delete.

Using forced-conflict synchronizations

No matter what type of consolidated database you are using, you can use MobiLink's forced-conflict mode to apply uploaded data to the consolidated database and handle rows with primary keys that match rows already in the consolidated database. This section describes an overview of the process.

Using your new script version name, create scripts for the following table events for each table (do not create scripts for any other table events):

- **begin_upload (Script Version = REBUILD)** use this script to create a temporary table in the consolidated database for the uploaded data.
- **upload_new_row_insert (Script Version = REBUILD)** use this script to insert the row into the temporary table. After all the rows have been inserted into the temporary table, the end_upload table event fires.
- **end_upload (Script Version = REBUILD)** use this script to move all the rows from the temporary table into the permanent table. For a row with a primary key value that is not already in the permanent table, insert the row. Otherwise, update the row in the permanent table (or use whatever conflict resolution you prefer). After all rows have been moved into the permanent table, drop the temporary table.

  Using an end_upload script instead of a resolve_conflict script lets you deal with all the rows uploaded into the temporary table at once. If 1000 rows were uploaded for a table, the end_upload event would be fired only one time, but resolve_conflict event would be fired 1000 times.

Example of using forced-conflict synchronizations

Assume that the following table is being synchronized from the remote database:

```
CREATE TABLE Product (
    prod_id             INTEGER NOT NULL,
    price               INTEGER,
    prod_name           VARCHAR(30),
    PRIMARY KEY( prod_id )
)
```

The corresponding table on the consolidated database often has an additional column (see below):

```
CREATE TABLE Product (
    prod_id             INTEGER NOT NULL,
    price               INTEGER,
    prod_name           VARCHAR(30),
    last_modified       DATETIME DEFAULT TIMESTAMP,
    PRIMARY KEY( prod_id )
)
```

The last_modified column is only used on the consolidated database. The purpose of the column is to identify whether a row has changed since a remote last downloaded the row. This prevents rows from being sent to a remote database if it already has the rows, and the rows have not changed. The download_cursor for the Product table uses a WHERE clause that references this column.

The Product table has the following events defined:

- begin_upload (Script Version = REBUILD)
- upload_new_row_insert (Script Version = REBUILD)
- end_upload (Script Version = REBUILD)

begin_upload (Script Version = REBUILD)

```
CREATE TABLE #Product (
    prod_id             INTEGER NOT NULL PRIMARY KEY,
    price               INTEGER,
    prod_name           VARCHAR(30),
    PRIMARY KEY( prod_id )
)
```

This begin_upload script creates the temporary table used to store all the rows uploaded from the remote database.

Note that this table does not have a last_modified column. The remote database does not have this column, so the temporary table that you upload rows into does not need this column.

The above syntax works with Microsoft SQL Server, Sybase Adaptive Server Enterprise, Oracle 8i, and Sybase Adaptive Server Anywhere. To use this technique with a different RDBMS, consult its documentation on how to create and reference a temporary table.

upload_new_row_insert (Script Version = REBUILD)

```
INSERT INTO #Product( prod_id, price, prod_name )
VALUES (?, ?, ?)
```

Note that this upload_new_row_insert script has been created for the Product table, but the script actually inserts into the temporary #Product table. This is one of the features of MobiLink: the names and column names do not have to match what is in the consolidated database. This has been done here for readability reasons. The order in which the columns are uploaded is what is important, not the column names.

end_upload (Script Version = REBUILD)

```
CALL sp_Product_insert_update();
DROP TABLE #Product
```

This end_upload script calls the sp_Product_insert_update stored procedure, which performs inserts and updates (see below).

Next, it deletes the temporary table. Now that the sp_Product_insert_update stored procedure has updated the real Product table in the consolidated database, the temporary table is no longer required.

Note that if you are using an Adaptive Server Anywhere consolidated database, you can use a global temporary table instead of the #Product table. This means you would replace the begin_upload event with a DELETE statement, and change the end_upload event to call only the stored procedure.

☞ For information about using global temporary tables, see “CREATE TABLE statement” in the *Adaptive Server Anywhere Reference Manual*.

That is it. These are all the events that you require for each table.

The stored procedure, sp_Product_insert_update, looks like this:

```
CREATE PROCEDURE sp_Product_insert_update()
BEGIN
     -- First insert all records that are in the temporary table
     -- that currently do NOT exist in the permanent table
     INSERT INTO Product( prod_id, price, prod_name )
     SELECT prod_id, price, prod_name
     FROM #Product t_p
     WHERE t_p.prod_id IS NOT IN (
          SELECT prod_id
          FROM Product p2
          WHERE p2.prod_id = t_p.prod_id
     );

     -- Second update all records that are in the Product table
     -- that currently DO exist in the #Product table
     -- with the data that is in the temporary table
     UPDATE Product
     SET price = t_p.price, prod_name = t_p.prod_name
     FROM #Product t_p
     WHERE Product.prod_id = t_p.prod_id;
END
```

You may wish to replace the second part of the stored procedure with your own business logic on how to handle uploads to rows that already exist in the consolidated database. In the above example, the uploaded row replaces the existing row.

On the remotes

Now that you have set up the consolidated database, you can set up the remote Adaptive Server Anywhere databases and upload their data. The MobiLink client for Adaptive Server Anywhere (*dbmlsync*) only uploads rows that have been changed since the last time the database synchronized. *dbmlsync* does this by scanning the Adaptive Server Anywhere transaction log for the tables that are involved in the publication (or the synchronization definition in version 7). In order for remote data to be uploaded, you have to ensure that the rows are in the remote database’s transaction log. This ensures that when *dbmlsync* next synchronizes, the data is uploaded (as inserts) to the consolidated database.

The simplest way to accomplish this is to rebuild the remote database using the dbunload utility with some special switches.

For MobiLink synchronization, remote Adaptive Server Anywhere databases must have the following:

1. A **publication**, to indicate which tables the remote should synchronize with the MobiLink server.

2. A **synchronization user**, to uniquely identify the remote database to the MobiLink server. This can be any value you decide on, but no two remote databases can have the same synchronization user name.

3. One or more **synchronization subscriptions**, to specify which publications the synchronization user synchronizes and to specify options for the synchronization. Typical options include the communications protocol, the host address for the MobiLink server, and the version name for the set of synchronization scripts you want to use.

If you are just beginning to use MobiLink synchronization, your remote databases do not yet have these items defined. If your remote databases have already been using MobiLink synchronization, then these items are already defined.

Example of rebuilding the remote database and uploading all data

Assume that the following publication, synchronization user, and synchronization subscription are being used in the remote database:

```
CREATE PUBLICATION Product(
        TABLE Product
 );

CREATE SYNCHRONIZATION USER store31;

CREATE SYNCHRONIZATION SUBSCRIPTION TO "Product"
    FOR store31
    TYPE ’tcpip’
    ADDRESS ’host=localhost;port=2439’
        OPTION ScriptVersion =’v1.0’;
```

The following steps rebuild the remote database and perform a synchronization so that all rows in the Product table are uploaded as inserts:

1. If you are recovering an existing consolidated database, remove and re-add the user to reset the synchronization progress value in the consolidated database.
   - When a remote database synchronizes, MobiLink stores the log offset that the remote Adaptive Server Anywhere database last successfully synchronized to. If you rebuild the Adaptive Server Anywhere database, this log offset is no longer valid.
   - Use the DBMLUSER utility to delete the user and then re-add the user to the MobiLink system tables (in the consolidated database).

   ```
   dbmluser -d -u store31 -c "dsn=cons"
   dbmluser -u store31 -c "dsn=cons"
   ```

2. Unload the schema of the remote database.
   - You need the schema to be separate from the data because you need the publications and subscription in place *before* you load the data back into the remote database. By doing this, DBMLSYNC uploads INSERTs to the consolidated database.

- Use the dbunload utility with the -n switch to unload only the schema:

  ```
  dbunload -y -c "dsn=sales" -n -r newsales.sql
  ```

3. Unload the data from the remote database.
   - Use the dbunload utility with the -d and -ix switches to unload only the data, and specify that the reload should be done *externally*. With an external reload, reloading of the data is recorded as INSERTs in the log file. Here is an example:

     ```
     dbunload -y -c "dsn=sales" -ix -d -r newsales_data.sql c:\unload
     ```

4. Create a new remote database and reload the schema into it:

   ```
   dbinit newsales.db
   dbisql -c dsn=newsales read newsales.sql
   ```

   This creates the schema again, as well as any publications, synchronization users, or subscriptions that were in place when you ran dbunload.

5. Next, you set up synchronization. The instructions you should follow depend on whether you already had synchronization settings in the remote database when you unloaded it.

   If you already had synchronization settings defined, then all you need to do is drop and redefine any synchronizations in order to reset the synchronization tracking information. For the example database, you can do this by logging into the newsales database and executing the following SQL commands. The CREATE SYNCHRONIZATION SUBSCRIPTION statement can be copied from your *newsales.sql* file.

   ```
   DROP SYNCHRONIZATION SUBSCRIPTION TO "Product"
               FOR store31;

   CREATE SYNCHRONIZATION SUBSCRIPTION TO "Product"
       FOR store31
       TYPE 'tcpip'
       ADDRESS 'host=localhost;port=2439'
           OPTION ScriptVersion ='v1.0';
   ```

   If you did not already have synchronization settings defined, then you need to define your normal publication, synchronization user, and synchronization subscription for the tables you want to synchronize. If you are repopulating a previously-existing consolidated database, then you may already have a publication.

   ```
   CREATE PUBLICATION Product(
       TABLE Product
   );

   CREATE SYNCHRONIZATION USER store31;

   CREATE SYNCHRONIZATION SUBSCRIPTION TO "Product"
       FOR store31
       TYPE 'tcpip'
       ADDRESS 'host=localhost;port=2439'
           OPTION ScriptVersion ='v1.0';
   ```

6. Reload the data into the remote database:

   ```
   dbisql -c "dsn=newsales" read newsales_data.sql
   ```

   Since the subscription was created in the previous step and you specified external reload when unloading the data out of the previous database, all the data is marked as yet-to-be-synchronized inserts.

7. Run *dbmlsync* to send the changes to the consolidated database. If you set up your consolidated database to have a different version of the synchronization scripts for the rebuild (for example, Script Version = REBUILD), then you can specify that version on the command line. This temporarily overrides the version specified in the synchronization subscription:

   ```
   dbmlsync -v -c "dsn=newsales" -o rem.txt -e "ScriptVersion=REBUILD"
   ```

8. Once *dbmlsync* completes in step 7, for any subsequent synchronizations, use the synchronization version from the subscription:

   ```
   dbmlsync -v -c dsn=newsales -o rem.txt
   ```

After the synchronization completes, all rows of the Purchase table in the remote database have been uploaded to the consolidated database.

You can repeat the above steps for each remote database. As described in the previous section, it is up to the synchronization logic in your consolidated database to handle any cases where uploaded rows have primary keys that match existing rows in the consolidated database. After performing these steps for each remote database, you have populated, or re-populated, the consolidated database.

Legal Notice