MobiLink Synchronization with High Availability Databases

Purpose
The purpose of this page is to discuss MobiLink synchronization with high availability databases.

Introduction
Data synchronization has become an important part of many corporate information systems. Synchronizing high availability databases and setting up synchronization systems without a single point of failure are two challenges that organizations face as they bring synchronization systems to the 24/7 requirements of modern data systems.

In data synchronization environments there needs to be a transport mechanism to facilitate data transfer from one database to another, and vice-versa. MobiLink synchronization provides a highly scalable session-based synchronization system that allows bi-directional synchronization between a main database, called the consolidated database, and many remote databases. The consolidated database can be one of several ODBC-compliant databases, including SQL Anywhere, HANA, Adaptive Server Enterprise, Oracle, Microsoft SQL Server, and IBM DB2. The image below provides a high level overview of the MobiLink synchronization system.

To create a synchronization system without a single point of failure, redundancy must be built in to each component of the system.

This document outlines how to configure MobiLink synchronization to run in a high availability environment using a SQL Anywhere 10 consolidated database with SQL Anywhere 10 remote databases. This document also shows how to remove single points of failure from a synchronizing environment.

Prerequisites
This document assumes you have the following:

- SQL Anywhere 10 (or higher) Database Server with High Availability Option
- SQL Anywhere 10 (or higher) MobiLink Server
- SQL Anywhere 10 (or higher) MobiLink Clients

Database Mirroring
Database mirroring is an availability and disaster-recovery technology for database management systems to ensure data and data...
synchronization is available any time. A database mirroring system is a configuration of either two or three database servers, each running on separate computers, which cooperate to maintain copies of a database. For SQL Anywhere, three database servers are required for failover in the mirroring system to be automatic.

Database mirroring consists of a primary server, mirror server, and arbiter server. The following image shows the communication between the three database servers. The primary server and mirror server each maintain a copy of the database files while the arbiter server helps determine which of the other two servers can take ownership of the database. The arbiter server does not maintain a copy of the database.

**High Availability Demonstration**

The image below illustrates the configuration of the high availability demonstration. All databases are SQL Anywhere running in high availability. All communication is using TCPIP.
High Availability for Consolidated Database

Database mirroring builds high availability into the consolidated database. The dotted line in the next image represents the failover of the MobiLink server if a role switch occurs in the database mirroring environment.

Consolidated Schema

The consolidated database schema consists of eight tables: Admin, Admin_del, Parent, Parent_del, Child, Child_del, Inventory, and Inventory_del. The tables that consist of *_del are delete tracking tables, specifically designed for monitoring the primary keys of deleted rows for synchronization purposes. Each table has a last_modified column and three triggers to maintain timestamp tracking which ensures only modified rows are synchronized to the remote database.

Eight tables are involved in synchronization: Admin, Parent, Child, Inventory, and their corresponding *_del tables. There are upload_insert, upload_delete, and upload_update table scripts written for each of the synchronizing tables to handle rows coming up from the remote database. There is a download_cursor to handle inserts and updates to send to remote databases, with the download_delete_cursor using the *_del tables to send deletes. This is an example of the Admin and Admin_del database schema:

```sql
/*----------------------------------------*/
/* Creating schema related to the table 'DBA.Admin'*/
*/
/* Create base Admin table */
CREATE TABLE Admin (  
    admin_id BIGINT DEFAULT GLOBAL AUTOINCREMENT PRIMARY KEY,  
    data VARCHAR(30),  
    last_modified TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
/* Create the shadow delete table 'DBA.Admin_del'. */
CREATE TABLE "DBA"."Admin_del" (  
    "admin_id" BIGINT NOT NULL,  
    "last_modified" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    PRIMARY KEY ("admin_id")
);
/* Create the shadow delete trigger 'Admin_ins'. */
CREATE TRIGGER "Admin_ins" AFTER INSERT ORDER 3  
ON "DBA"."Admin"  
REFERENCING  
NEw AS inserted  
FOR EACH STATEMENT
BEGIN
```
/*
 * Delete the row from the shadow delete table. (This trigger is only needed if deleted * primary keys can be re-inserted.)
 */
DELETE FROM "DBA"."Admin_del"
WHERE EXISTS (
  SELECT 1
  FROM inserted
  WHERE "DBA"."Admin_del"."admin_id" = inserted."admin_id"
);
END;

/* Create the shadow delete trigger 'Admin_dlt'. */
CREATE TRIGGER "Admin_dlt" AFTER DELETE ORDER 2
ON "DBA"."Admin"
REFERENCING
OLD AS deleted
FOR EACH STATEMENT
BEGIN
/* Insert the row into the shadow delete table. */
INSERT INTO "DBA"."Admin_del" ( "admin_id", "last_modified" )
SELECT deleted."admin_id", CURRENT_TIMESTAMP
FROM deleted;
END;

/* Create the timestamp trigger 'Admin_upd'. */
CREATE TRIGGER "Admin_upd" AFTER UPDATE
OF "admin_id", "data"
ORDER 1 ON "DBA"."Admin"
REFERENCING NEW AS inserted OLD AS deleted
FOR EACH STATEMENT
BEGIN
/* Update the column last_modified in modified row. */
UPDATE "DBA"."Admin"
SET "last_modified" = CURRENT_TIMESTAMP
FROM inserted
WHERE "DBA"."Admin"."admin_id" = inserted."admin_id";
END;

/* Create an index for the 'download_cursor' script. */
CREATE INDEX "Admin_ml" ON "DBA"."Admin" ( "last_modified" );

Below is an example of the upload_insert table script and the download_cursor for the Admin and Admin_del tables. The upload_insert table script is used to generate new inserts that are then sent up to the consolidated database. The download_cursor is used to generate upsert statements, which are a combination of insert and update statements that are then sent to the remote databases. To create the synchronization scripts, the ml_add_table_script MobiLink procedure is called:

CALL ml_add_table_script (
  'https_v1',
  'Admin',
  'upload_insert',
  /* Insert the row into the consolidated database. */
  INSERT INTO "DBA"."Admin" ( "admin_id", "data" )
  VALUES ( {ml r."admin_id"}, {ml r."data"} )
);

CALL ml_add_table_script (
  'https_v1',
  'Admin',
  'download_cursor',
  'SELECT "DBA"."Admin"."admin_id",
       "DBA"."Admin"."data"
  FROM "DBA"."Admin"
  WHERE "DBA"."Admin"."last_modified" >= {ml s.last_table_download}
);

In SQL Anywhere 12 and below, in order to use named parameters in the synchronization scripts, the column names must be added by calling the ml_add_column MobiLink system procedure. This is how to add the column names so they can be used as named parameters in the synchronization scripts:

CALL ml_add_column (  'https_v1',  'Admin', null, null );

CALL ml_add_column (}
'https_v1',
'Admin',
'admin_id',
'bigint');
CALL ml_add_column {
'https_v1',
'Admin',
'data',
'varchar');

Starting the Consolidated Database in High Availability (SQL Anywhere 10)

See the documentation or the samples directory (Samples\SQLAnywhere\DBMirror) for a full tutorial on how to set up High Availability on SQL Anywhere 12 and SQL Anywhere 16. The instructions for setting up high-availability with SQL Anywhere 10 follow below.

Arbiter Server

The arbiter server does not maintain a copy of the consolidated database, but it is required for determining which server will become the primary server, both during startup and during failover. To start the arbiter server:

```bash
"%SQLANY10%\win32\dbspawn" -f "%SQLANY10%\win32\dbsrv10" -x tcpip(port=2637) -n arbiter -xa AUTH=abc;DBN=MLcons
-f arbiterstate.txt -su sql
```

- `dbspawn` is a utility that starts the database server as a process in the background
- `-f` specifies that another database server should be started if the default server is running
- `dbsrv10` is the database server
- `-x tcpip(port=2637)` specifies the database server to listen using TCPIP on port 2637
- `-n arbiter` specifies the database server name as arbiter
- `-xa` specifies the database names and authentication strings for the arbiter server
- `AUTH=abc` specifies the arbiter authentication string as abc
- `DBN=MLcons` specifies the database name as MLcons
- `-xf arbiterstate.txt` specifies the arbiterstate.txt file that will be used to maintain state information about the database mirror system
- `-su sql` specifies the password for the DBA user of the utility database

Server 1 (Primary)

The primary server will maintain the active copy of the database file and transaction logs. This server will accept active connections and process database transactions. All transaction will be sent to the mirror server by the primary server. To start server 1:

```bash
"%SQLANY10%\win32\dbspawn" -f "%SQLANY10%\win32\dbsrv10" -x tcpip(PORT=2638) -xf server1state.txt -su sql
-server1 .\server1\MLcons.db -n MLcons -sn tcpip_cons -xp partner=(ENG=server2;LINKS=tcpip(PORT=2639;TIMEOUT=1));auth=abc;arbiter=(ENG=arbiter;LINKS=tcpip(PORT=2637;TIMEOUT=1));mode=sync
```

- `dbspawn` is a utility that starts the database server as a process in the background
- `-f` specifies that another database server should be started if the default server is running
- `dbsrv10` is the database server
- `-x tcpip(PORT=2638)` specifies the database server to listen using TCPIP on port 2638
- `-xf server1state.txt` specifies the server1state.txt file will be used to maintain state information about the database mirror system
- `-su sql` specifies the password for the DBA user of the utility database
- `-n server1` specifies the database server name as server1
- `.\server1\MLcons.db` specifies the location of the database file
- `-n MLcons` specifies the database name as MLcons
- `-sn tcpip_cons` specifies the alternative database server name as tcpip_cons
- `-xp partner` specifies the location of the partner server for database mirroring
- `-xp arbiter` specifies the location of the arbiter server for database mirroring
- `-xp auth` specifies the authentication string for the arbiter server
- `-xp mode` specifies the type of database mirroring
- `ENG=` specifies the database engine name
- `LINKS=` specifies the communication protocols
- `PORT=` specifies the listening port
- `TIMEOUT=` specifies the length of time to wait for a response when establishing communications

Server 2 (Mirror)

The mirror server does not accept active database connections. This mirror server receives all transactions that have been applied to the primary
server directly from the primary server. The mirror server will become the primary server if a role switch occurs. To start server 2:

```
"%SQLANY10%\win32\dbspawn" -f "%SQLANY10%\win32\dbsrv10" -x tcpip(port=2639) -xf server2state.txt -su sql
    -n server2 .\server2\MLcons.db -n MLcons -an tcpip_cons -xp partner=(ENG=server1;LINKS=tcpip(PORT=2638;TIMEOUT=1));auth=abc;arbiter=(ENG=arbiter;LINKS=tcpip(PORT=2637;TIMEOUT=1));mode=sync
```

- `dbspawn` is a utility that starts the database server as a process in the background
- `-f` specifies that another database server should be started if the default server is running
- `dbsrv10` is the database server
- `-x tcpip(port=2639)` specifies the database server to listen using TCPIP on port 2639
- `-xf server2state.txt` specifies the server2state.txt file will be used to maintain state information about the database mirror system
- `-su sql` specifies the password for the DBA user of the utility database
- `-n server2` specifies the database server name as server2
- `.\server2\MLcons.db` specifies the location of the database file
- `-n MLcons` specifies the database name as MLcons
- `-an tcpip_cons` specifies the alternative database server name as tcpip_cons
- `-xp partner` specifies the location of the partner server for database mirroring
- `-xp arbiter` specifies the location of the arbiter server for database mirroring
- `-xp auth` specifies the authentication string for the arbiter server
- `-xp mode` specifies the type of database mirroring
- `ENG=` specifies the database engine name
- `LINKS=` specifies the communication protocols
- `PORT=` specifies the listening port
- `TIMEOUT=` specifies the length of time to wait for a response when establishing communications

### Starting the MobiLink Server

The MobiLink server establishes an ODBC connection with the primary server and will process the upload stream from the MobiLink client and generate the download stream to be sent to the clients. To start the MobiLink server:

```
start "" "%SQLANY16%\bin64\mlsrv16" -o mlsrv.out -zs ml_tcpip -c "DSN=tcpip_cons" -dl -zu+ -x tcpip(port=2439)
```

- `start ""` is a Windows command prompt program that starts a program in a separate window
- `mlsrv16` is the MobiLink server executable
- `-o mlsrv.out` specifies mlsrv.out file will contain the MobiLink server output
- `-zs ml_tcpip` specifies the MobiLink server name will be ml_tcpip
- `-c` specifies the ODBC database connection parameters
- `DSN=tcip_cons` specifies that the data source tcpip_cons be used to connect to the consolidated database
- `-dl` specifies that all messages be displayed on the MobiLink console
- `-zu+` specifies that new MobiLink users be added automatically when the authenticate_user script is undefined
- `-x tcpip(port=2439)` specifies the MobiLink server to listen on port 2439

To connect MobiLink to the high availability system, the DSN "tcip_cons" should look similar to the following screen with multiple host addresses defined to target the two systems. The server name should be the shared or "alternate name" (-sn) of the two SQL Anywhere database servers.

### Note:
If you are using the SQL Anywhere 12 or SQL Anywhere 16 tutorial, the alternate server name used in the tutorial is mirror_de mo_primary
High Availability for Remote Databases

A database mirroring environment can be used for configuring the remote database system, but initiating data synchronization is more complex. Data synchronization is initiated when the MobiLink client establishes a connection with the remote database and the MobiLink server. The MobiLink client needs to scan the active transaction log and offline transaction logs. However, in a database mirroring environment it is not always known which database is acting as the primary server and where the active and offline transaction logs exist. When starting the MobiLink client, this needs to be considered. The image below shows how the MobiLink client needs to connect with the primary database server.
Remote Databases Schema

The remote databases schema consists of four tables: Admin, Parent, Child, and Inventory. The four tables consist of the following schema:

```sql
/*-----------------------------------------------*/
/* Create table 'DBA.Admin' */
CREATE TABLE "DBA"."Admin" (  
    "admin_id" bigint default global autoincrement primary key,  
    "data" varchar(30) null  
);
/*-----------------------------------------------*/
/* Create table 'DBA.Child' */
CREATE TABLE "DBA"."Child" (  
    "child_id" bigint default global autoincrement primary key,  
    "parent_id" bigint null,  
    "data" varchar(30) null  
);
/*-----------------------------------------------*/
/* Create table 'DBA.Inventory' */
CREATE TABLE "DBA"."Inventory" (  
    "inventory_id" bigint default global autoincrement primary key,  
    "data" integer null  
);
/*-----------------------------------------------*/
/* Create table 'DBA.Parent' */
CREATE TABLE "DBA"."Parent" (  
    "parent_id" bigint default global autoincrement primary key,  
    "user_id" varchar(128) null,  
    "data" varchar(30) null  
);
/*-----------------------------------------------*/
```

Each remote database has a unique Global_database_id set using the following SQL statement:

```sql
SET OPTION PUBLIC.Global_database_id = 123456789;
```

The MobiLink publication includes all four tables in the definition and the synchronizing user is defined as remN, where N is the value of the remote. For example, the publication is defined as follows:

```sql
/*-----------------------------------------------*/
/* Create publication 'pub1' */
CREATE PUBLICATION "pub1" (  
    /* ... */  
);
/*-----------------------------------------------*/
```
The synchronizing user is defined as follows:

```sql
CREATE SYNCHRONIZATION USER "rem100";
```

Finally, a synchronization subscription is defined to associate the publication with the synchronizing user. The synchronization subscription is defined as follows:

```sql
CREATE SYNCHRONIZATION SUBSCRIPTION TO "pub1" FOR "rem100" OPTIONS ScriptVersion='https_v1';
```

### Starting the Remote Database in High Availability (SQL Anywhere 10 Instructions)

For a detailed explanation of each component involved in starting the database server in high availability, see the breakdown in the Starting the Consolidated Database in High Availability section.

#### Arbiter

To start the arbiter server:

```sh
"%SQLANY10\win32\dbspawn" -f "%SQLANY10\win32\dbsrv10" -x tcpip(port=2700) -n rem_arbiter -xa AUTH=abc;DBN=MLrem -xf rem_arbiterstate.txt -su sql
```

#### Server 1 (Primary)

To start server 1:

```sh
"%SQLANY10\win32\dbspawn" -f "%SQLANY10\win32\dbsrv10" -x tcpip(PORT=2701) -xf rem_server1state.txt -su sql
  -n rem_server1 .\rem_server1\MLrem.db -n MLrem -sn tcpip_rem -xp partner=(ENG=rem_server2;LINKS=tcpip(PORT=2702;TIMEOUT=1));auth=abc;arbiter=(ENG=rem_arbiter;LINKS=tcpip(PORT=2700;TIMEOUT=1));mode=synca
```

#### Server 2 (Mirror)

To start server 2:

```sh
"%SQLANY10\win32\dbspawn" -f "%SQLANY10\win32\dbsrv10" -x tcpip(port=2702) -xf rem_server2state.txt -su sql
  -n rem_server2 .\rem_server2\MLrem.db -n MLrem -sn tcpip_rem -xp partner=(ENG=rem_server1;LINKS=tcpip(PORT=2701;TIMEOUT=1));auth=abc;arbiter=(ENG=rem_arbiter;LINKS=tcpip(PORT=2700;TIMEOUT=1));mode=sync
```

### Starting the MobiLink Client

The MobiLink client will generate the upload stream from the remote database's transaction log to send to the MobiLink server. The client will then process and apply the download stream. To start the MobiLink client:

```sh
"%SQLANY16\bin64\dbmlsync" -k -c "DSN=tcpip_rem" -mp sql -v+ -o rem.txt -e "scn=on;ctp=tcpip;adr=''timeout=1000;host=localhost;port=2439''"
```

- `dbmlsync` is the MobiLink client
- `-k` specifies that the client window be closed after synchronization has completed
- `-c` specifies the connection parameters for the remote database
- `DSN=tcpip_rem` specifies that the data source tcpip_rem be used to connect to the consolidated database
- `-mp` specifies the MobiLink password
- `-v+` specifies verbose logging of the client
- `-o rem.txt` specifies the rem.txt file will contain the output of the client console
- `-e` specifies the extended options for the MobiLink client
- `--cscn` specifies that column names are to be sent in the upload stream
- `--ctp` specifies the communication type for connecting to the MobiLink server
- `--adr` specifies the address of the MobiLink server
- `timeout` specifies the amount of time a client waits for a network operation before giving up
- `host` specifies the host name where the MobiLink server is running
- `port` specifies the port number the MobiLink server is listening on for new connections

If there are offline transaction logs associated with the database, this is specified using the `--dir=<path>` extended MobiLink client option.

**Removing a Single Point of Failure with the MobiLink Server**

Configuring a database mirroring environment still leaves a single point a failure in the data synchronization system. The next figure shows a single point of failure with the MobiLink server. Using multiple MobiLink servers and load balancers (failover cluster) or the Relay Server can remove the single point of failure in the system. The dotted lines in figure 6 represents the failover of the server if a role switch occurs in the database mirroring environment, or if there is a failure of MobiLink server #1.

With the introduction of load balancers (failover cluster) and multiple MobiLink servers, each component of the data synchronization environment is redundant. The data synchronization system will now function even if hardware or software failures occur in the environment.

**Removing a Single Point of Failure with MobiLink Clients**

In a database mirroring system, at least two machines must fail before the entire synchronization system fails. However in the next diagram, if the machine running the MobiLink client fails then synchronization fails. The MobiLink client introduces a single point of failure. To avoid this single point of failure, an implementation using an external procedure to call the MobiLink client can be triggered via a scheduled event that is stored in both the primary and mirror databases.
This prevents the need for an external application to initiate data synchronization and if a role switch occurs, synchronization will still be initiated by the remote database. Scheduled events are a feature of SQL Anywhere that allows logic to be triggered based on a predefined schedule. The event will only fire on the server that is currently the primary server.

Here is an example of a scheduled event to call the external procedure:

```sql
CREATE EVENT SyncEvent
SCHEDULE daily_sync
BETWEEN '5:00AM' AND '11:00PM'
EVERY 1 MINUTES
HANDLER
BEGIN
    MESSAGE 'Synchronization Triggered';
    IF CAST( EVENT_PARAMETER( 'NumActive' ) AS INTEGER ) > 1 THEN RETURN;
    END IF;
    CALL xp_cmdshell( 'dbmlsync -k -c "DSN=tcpip_rem" -mp sql -v+ -o rem.txt -e
    "scn=on;ctp=tcpip;adr='"timeout=1000;host=localhost;port=2439'''"' , 'no_output' );
END;
```

In this statement, the event parameter NumActive is used to ensure that an instance of synchronization is not already running. Synchronization is scheduled to occur every 10 minutes during business hours.

**Summary**

Corporations are going to continue to move towards always available data and occasionally connected applications. This will continue to increase the importance of enterprise database management systems to remain functional and online in 24/7 environments.

Database mirroring is an effective way to provide database availability and disaster-recovery for data synchronization environments. Failover of database mirroring environments is both fast and automatic, providing seamless, reliable service and coupled with initiating synchronization from the remote database using the xp_cmdshell in a scheduled event, will eliminate single points of failure in the mirrored architecture.

**Related Content**

**Related Documents**

**Related SAP Notes/KBAs**