CHAPTER 11

Administering SQL Remote for Adaptive Server Anywhere

About this chapter

This chapter details set-up, and management issues for SQL Remote administrators using Adaptive Server Anywhere as a consolidated database.

Contents

Running the Message Agent

This section describes how to run the Message Agent for Adaptive Server Anywhere.

☞ For information on features of the Message Agent that are common to Adaptive Server Anywhere and Adaptive Server Enterprise, see "SQL Remote Administration" on page 217.

Starting the Message Agent

The Message Agent has a set of command-line switches that control its behavior. The only command-line switch that is required for the Message Agent to run is the connection parameters switch (`-c`).

The connection parameters are described in the chapter "Connecting to a Database" in the *Adaptive Server Anywhere User's Guide*.

Verbose keyword	Short form	Argument
DatabaseFile	DBF	string
DatabaseName	DBN	string
DatabaseSwitches	DBS	string
EngineName	ENG	string
Password	PWD	string
Start	Start	string
Userid	UID	string

Running the Message Agent as a service

If you are running the Message Agent in continuous mode (not batch mode), on Windows NT or Windows 95, you may wish to keep the Message Agent running all the time that the server is running.

You can do this by running the Message Agent as a **service** under Windows NT or Windows 95. A service for Windows NT can be configured to keep running even when the current user logs out, and to start as soon as the operating system is started.

☞ For a full description of running programs as services, see the chapter "Running the Database Server" in the *Adaptive Server Anywhere User's Guide*.

The Message Agent and replication security

In the tutorials in the previous chapter, the Message Agent was run using a user ID with DBA permissions. The operations in the messages are carried out from the user ID specified in the Message Agent connection string; by using the user ID **DBA**, you can be sure that the user has permissions to make all the changes.

In many situations, distributing the DBA user ID and password to all remote database users is an unacceptable practice for security and data privacy reasons. SQL Remote provides a solution that enables the Message Agent to have full access to the database in order to make any changes contained in the messages without creating security problems.

A special permission, REMOTE DBA, has the following properties:

- **No distinct permissions when not connected from the Message Agent** A user ID granted REMOTE DBA authority has no extra privileges on any connection apart from the Message Agent. Therefore, even if the user ID and password for a REMOTE DBA user is widely distributed, there is no security problem. As long as the user ID has no permissions beyond CONNECT granted on the database, no one can use this user ID to access data in the database.
- **Full DBA permissions from the Message Agent** When connecting from the Message Agent, a user ID with REMOTE DBA authority has full DBA permissions on the database.

Using REMOTE DBA permission

A suggested practice is to grant REMOTE DBA authority at the consolidated database to the publisher and to each remote user. When the remote database is extracted, the remote user becomes the publisher of the remote database, and is granted the same permissions they were granted on the consolidated database, including the REMOTE DBA authority which enables them to use this user ID in the Message Agent connection string. Adopting this procedure means that there are no extra user IDs to administer, and each remote user needs to know only one user ID to connect to the database, whether from the Message Agent (which then has full DBA authority) or from any other client application (in which case the REMOTE DBA authority grants them no extra permissions).

Granting REMOTE DBA permission

You can grant REMOTE DBA permissions to a user ID named **dbremote** as follows:

```
GRANT REMOTE DBA
TO dbremote
IDENTIFIED BY dbremote
```

In Adaptive Server Anywhere, you can add the REMOTE DBA authority to a remote user by checking the appropriate option in the New Remote User Wizard.

Error reporting and handling

This section describes how errors are reported and handled by the Message Agent.

Default error handling

The default action taken by the Message Agent when an error occurs is to record the fact in its log output. The Message Agent sends log output to a window or a log file recording its operation. By default, log output is sent to the window only; the -o command-line option sends output to a log file as well.

The Message Agent log includes the following:

- Listing of messages applied.
- Listing of failed SQL statements.
- Listing of other errors.

UPDATE conflicts are not errors

UPDATE conflicts are not errors, and so are not reported in the Message Agent output.

Ignoring errors

There may be exceptional cases where you wish to allow an error encountered by the Message Agent when applying SQL statements to go unreported. This may arise when you know the conditions under which the error occurs and are sure that it does not produce inconsistent data and that its consequences can safely be ignored.

To allow errors to go unreported, you can create a BEFORE trigger on the action that causes the known error. The trigger should signal the REMOTE_STATEMENT_FAILED SQLSTATE (5RW09) or SQLCODE (-288) value.

For example, if you wish to quietly fail INSERT statements on a table that fail because of a missing referenced column, you could create a BEFORE INSERT trigger that signals the REMOTE_STATEMENT_FAILED SQLSTATE when the referenced column does not exist. The INSERT statement fails, but the failure is not reported in the Message Agent log.

Implementing error handling procedures

SQL Remote allows you to carry out some other process in addition to logging a message if an error occurs. The Replication_error database option allows you to specify a stored procedure to be called by the Message Agent when an error occurs. By default no procedure is called.

The procedure must have a single argument of type CHAR, VARCHAR, or LONG VARCHAR. The procedure is called twice: once with the error message and once with the SQL statement that causes the error.

While the option allows you to track and monitor errors in replication, you must still design them out of your setup: this option is not intended to resolve such errors.

For example, the procedure could insert the errors into a table with the current time and remote user ID, and this information can then replicate back to the consolidated database. An application at the consolidated database can create a report or send e-mail to an administrator when errors show up.

☞ For information on setting the REPLICATION_ERROR option, see "SQL Remote options" on page 323.

Example: e-mailing notification of errors

You may wish to receive some notification at the consolidated database when the Message Agent encounters errors. This section demonstrates a method to send Email messages to an administrator when an error occurs.

A stored procedure

The stored procedure for this example is called **sp_LogReplicationError**, and is owned by the user **cons**. To cause this procedure to be called in the event of an error, set the **Replication_error** database option using Interactive SQL or Sybase Central:

```
SET OPTION PUBLIC.Replication_error =
   'cons.sp_LogReplicationError'
```

The following stored procedure implements this notification:

```
CREATE PROCEDURE cons.sp_LogReplicationError
   (IN error_text LONG VARCHAR)
BEGIN
   DECLARE current_remote_user CHAR(255);
   SET current_remote_user = CURRENT REMOTE USER;

   // Log the error
   INSERT INTO cons.replication_audit
      ( remoteuser, errormsg)
   VALUES
      ( current_remote_user, error_text);
```

```
   COMMIT WORK;

   //Now notify the dba an error has occurred
   // using email. We only want this information if  //
the error occurred on the consolidated database
   // We want the email to contain the error strings //
the Message Agent is passing to the procedure
   IF CURRENT PUBLISHER = 'cons' THEN
      CALL sp_notify_dba( error_text );
   END IF
END;
```

The stored procedure calls another stored procedure to manage the sending of Email:

```
CREATE PROCEDURE sp_notify_dba(in msg long varchar)
BEGIN
   DECLARE rc INTEGER;
   rc=call xp_startmail(mail_user='davidf');
   //If successful logon to mail
   IF rc=0 THEN
   rc=call xp_sendmail(
         recipient='Doe, John; John, Elton',
         subject='SQL Remote Error',
         "message"=msg);
   //If mail sent successfully, stop
      IF rc=0 THEN
         call xp_stopmail()
      END IF
   END IF
END;
```

An audit table

The audit table is as follows:

```
CREATE TABLE replication_audit (
   id    INTEGER DEFAULT AUTOINCREMENT,
   pub   CHAR(30) DEFAULT CURRENT PUBLISHER,
   remoteuser   CHAR(30),
   errormsg  LONG VARCHAR,
   timestamp    DATETIME DEFAULT CURRENT TIMESTAMP,
   PRIMARY KEY (id,pub)
);
```

The columns have the following meaning:

Column	Description
pub	Current publisher of the database (lets you know at what database it was inserted)
remoteuser	Remote user applying the message (lets you know what database it came from)
errormsg	Error message passed to the Replication_error procedure

Here is a sample insert into the table from the above error:

```
INSERT INTO cons.replication_audit
   (  id,
      pub,
      remoteuser,
      errormsg,
      "timestamp")
VALUES
   (  1,
      'cons',
      'sales',
      'primary key for table ''reptable'' is not unique
(-193)',
      '1997/apr/21 16:03:13.836')
COMMIT WORK
```

Since Adaptive Server Anywhere supports calling external DLLs from stored procedures you can also design a paging system, instead of using Email.

An example of an error

For example, if a row is inserted at the consolidated using the same primary key as one inserted at the remote, the Message Agent displays the following errors:

> Received message from "cons" (0-0000000000-0)
>
> SQL statement failed: (-193) primary key for table 'reptable' is not unique
>
> INSERT INTO cons.reptable(id,text,last_contact)
>
> VALUES (2,'dave','1997/apr/21 16:02:38.325')
>
> COMMIT WORK

The messages that arrived in Doe, John and Elton, John's email each had a subject of SQL Remote Error:

> primary key for table 'reptable' is not unique (-193)
>
> INSERT INTO cons.reptable(id,text,last_contact) VALUES (2,'dave','1997/apr/21 16:02:52.605')

Transaction log and backup management

The importance of good backup practices

Replication depends on access to operations in the transaction log, and access to old transaction logs is sometimes required. This section describes how to set up backup procedures at the consolidated and remote databases to ensure proper access to old transaction logs.

It is crucial to have good backup practices at SQL Remote consolidated database sites. A lost transaction log could easily mean having to re-extract remote users. At the consolidated database site, a transaction log mirror is recommended.

For information on transaction log mirrors and other backup procedure information, see the chapter "Backup and Recovery", in the *Adaptive Server Anywhere User's Guide*.

Ensuring access to old transactions

All transaction logs must be guaranteed available until they are no longer needed by the replication system.

In many setups, users of remote databases may receive updates from the office server every day or so. If some messages get lost or deleted, and have to be resent by the message-tracking system, it is possible that changes made several days ago will be required. If a remote user takes a vacation, and messages have been lost in the meantime, changes weeks old may be required. If the transaction log is backed up daily, the log with the changes will no longer be running on the server.

Setting the transaction log directory

When the Message Agent needs to scan transaction logs other than the current log, it looks through all the transaction log files kept in a designated **transaction log directory**. A setting on the Message Agent command line tells the Message Agent which directory this is.

Example

For example, the following command line tells the Message Agent to look in the directory *e:\archive* to find old transaction logs. The command must be entered all on one line.

```
dbremote -c "eng=server_name;uid=dba;pwd=sql" e:\archive
```

Log names are not important

The Message Agent opens all the files in the transaction log directory to determine which files are logs, so the actual names of the log files are not important.

This section describes how you can set up a backup procedure to ensure that such a directory is kept in proper shape.

Backup utility options

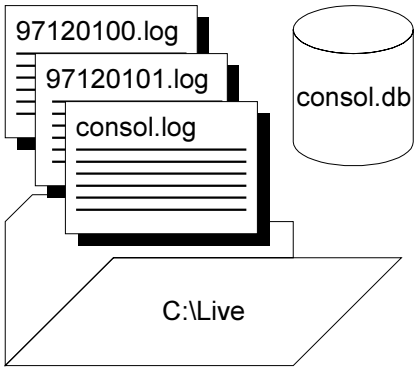
The Adaptive Server Anywhere backup utility has several options, accessible through Sybase Central wizard selections or through *dbbackup* command-line switches, that control its behavior.

This section describes two approaches to using the backup utility in SQL Remote consolidated database backups. Backups must ensure that a set of transaction logs suitable for use by the Message Agent is always available.

Using the live directory as the transaction log directory

It is recommended that you use the option to rename and restart the transaction log when backing up the consolidated database and remote database transaction logs. For the *dbbackup* command-line utility, this is the `-r` command-line switch.

The figure below illustrates a database named *consol.db*, with a transaction log named *consol.log* in the same directory. For the sake of simplicity, we consider the log to be in the same directory as the database, although this would not be generally safe practice in a production environment. The directory is named *c:\live*.



A backup command line

The following command line backs up the database using the rename and restart option:

```
dbbackup -r -c "uid=dba;pwd=sql" c:\archive
```

The connection string options would be different for each database.

Effects of the backup

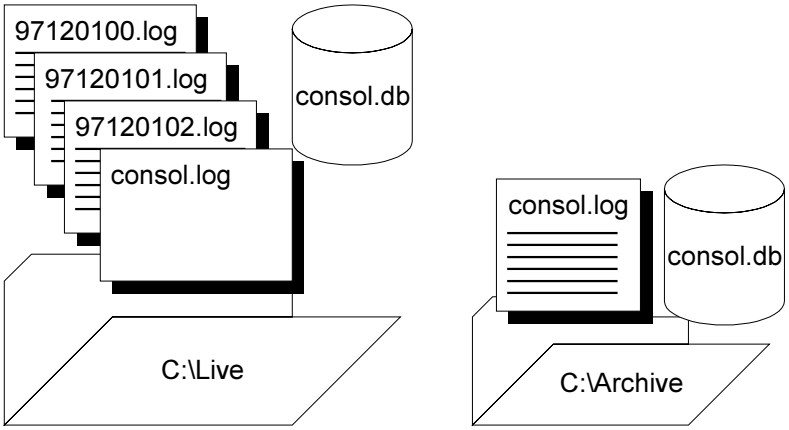
If you back up the transaction log to a directory *c:\archive* using the rename and restart option, the Backup utility carries out the following tasks:

1 Backs up the transaction log file, creating a backup file *c:\archive\consol.log*.

2 Renames the existing transaction log file to *971201nn.log*, where *nn* is the lowest available integer, starting at *00*.

3 Starts a new transaction log, as *consol.log*.

After several backups, the live directory contains a set of sequential transaction logs.



A Message Agent command line

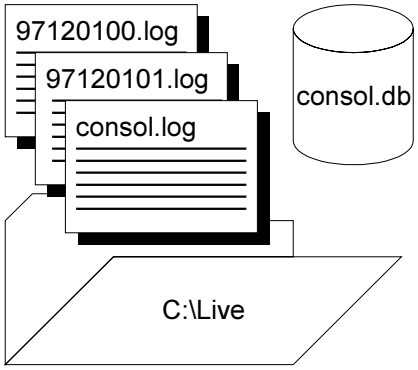
You can run the Message Agent with access to these log files using the following command line:

```
dbremote -c "dbn=hq;..." c:\live
```

Using the backup directory as the transaction log directory

An alternative procedure is to use the backup directory as the transaction log directory.

Again, the figure below illustrates a database named *consol.db*, with a transaction log named *consol.log* in the same directory. For the sake of simplicity, we consider the log to be in the same directory as the database, although this would not be generally safe practice in a production environment. The directory is named *c:\live*.



A backup command line

The following command line backs up the database using the rename and restart option, and also uses an option to rename the transaction log backup file:

```
dbbackup -r -k -c "uid=dba;pwd=sql" c:\archive
```

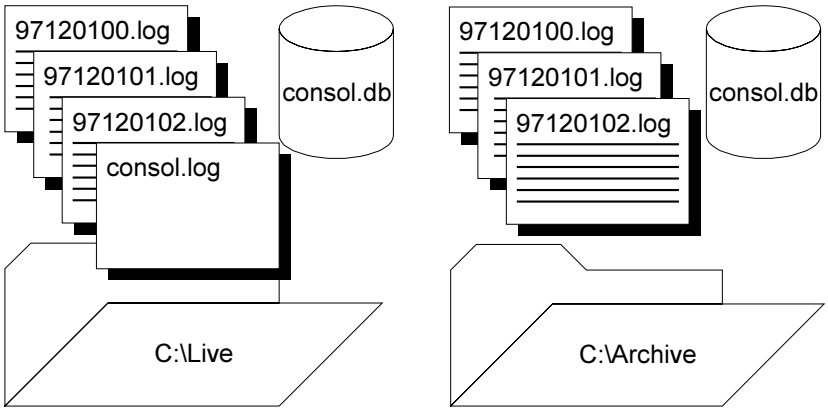
The connection string options would be different for each database.

Effects of the backup

If you back up the transaction log to a directory *c:\archive* using the rename and restart option and the log renaming option, the Backup utility carries out the following tasks:

1. Renames the existing transaction log file to *971201nn.log*, where *nn* is the lowest available integer, starting at *00*.
2. Backs up the transaction log file to the backup directory, creating a backup file named *971201nn.log*
3. Starts a new transaction log, as *consol.log*.

After several backups, the live directory and also the archive directory contain a set of sequential transaction logs.



A Message Agent command line

You can run the Message Agent with access to these log files using the following command line:

```
dbremote -c "dbn=hq;..." c:\archive
```

> **Old log names different before 5.5.01**
> Prior to release 5.5.01 of Adaptive Server Anywhere, the old log files were named *consol.l00*, *consol.l01*, and so on. The name change was introduced to allow more old logs to be stored. As the Message Agent scans all the files in the specified directory, regardless of their names, the name change should not affect existing applications.

Managing old transaction logs

All transaction logs must be guaranteed available until they are no longer needed by the replication system: at that point, they can be discarded.

The replication system no longer needs the logs when all remote databases have received and successfully applied the messages contained in the log files. Remote databases confirm the successful receipt of messages from the consolidated database, and the confirmation sets a value in the consolidated database SQL Remote tables (see "The message tracking system" on page 252). The old transaction logs at the consolidated database are no longer needed by SQL Remote when this receipt confirmation has been received from all remote databases.

Using the Delete_old_logs option

You can use the Delete_old_logs database option at the consolidated database to manage old transaction logs automatically.

The DELETE_OLD_LOGS database option is set by default to OFF. If it is set to ON, then the old transaction logs are deleted automatically by the Message Agent when they are no longer needed. A log is no longer needed when all subscribers have confirmed receiving all changes recorded in that log file.

You can set the DELETE_OLD_LOGS option either for the PUBLIC group or just for the user contained in the Message Agent connection string.

Example

- The following statement sets the public DELETE_OLD_LOGS:

```
SET OPTION PUBLIC.DELETE_OLD_LOGS = 'ON'
```

Recovery from database media failure for consolidated databases

This section describes how to recover from a media failure on the database device at the consolidated database.

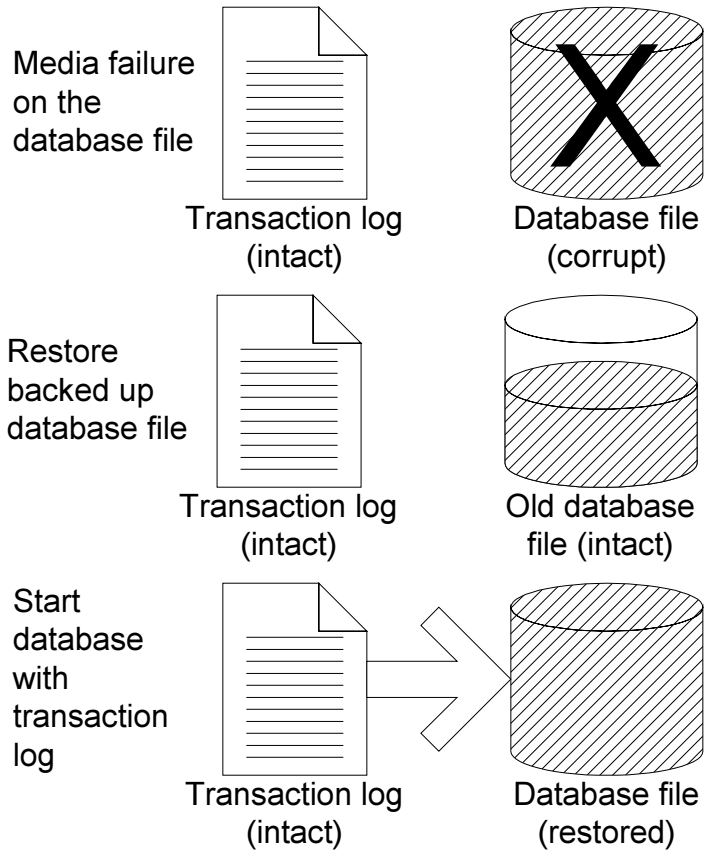
The procedures to follow are easiest to describe if there is only one transaction log file. While this might not be common for consolidated databases, it is described first, followed by a more common but complicated situation with a set of transaction log files.

Recovery with a single transaction log

In this case, we assume that there is a single transaction log file, which has existed since the database was created. Also, we assume previous backups of the database file have been made and are available, for example on tape.

❖ **To recover the database:**

1. Make a copy of the database and log file.
2. Restore the database (*.db*) file, *not* the log file, from tape into a temporary directory.
3. Start the database using the existing transaction log and the `-a` command-line switch, to apply the transactions and bring the database file up to date.
4. Start the database in your normal way. Any new activity will be appended to the current transaction log.



Example

This example illustrates recovery using a mirrored transaction log.

Suppose you have a consolidated database file named *consol.db* in a directory *c:\dbdir*, and a transaction log file *c:\logdir\consol.log* which is mirrored to *d:\mirdir\consol.mlg*.

❖ **To recover from media failure on the C drive:**

1. Backup the mirrored transaction log *d:\mirdir\consol.mlg*.
2. Replace the failed hardware and re-install all affected software.
3. Create a temporary directory to perform the recovery in (for example, *c:\recover*)
4. Restore the most recent backup of the database file, *consol.db*, to *c:\recover\consol.db*.
5. Copy the mirror transaction log, *d:\mirdir\consol.mlg*, to the recovery directory with a *.log* extension, giving *c:\recover\consol.log*.

6 Start the database using the following command line:

```
dbeng6 -a C:\RECOVER\CONSOL.DB
```

7 Shutdown the database server.

8 Backup the recovered database and transaction log from *c:\recover*.

9 Copy the files from *c:\recover* to the appropriate production directories:

- Copy *c:\recover\consol.db* to *c:\dbdir\consol.db*
- Copy *c:\recover\consol.log* to *c:\dbdir\consol.lOG*, and to *d:\mirdir\consol.mlg*.

10 Restart your system normally.

Recovery with multiple transaction logs

If you have a set of transaction logs, the procedure is different. We assume previous backups of the database file have been made and are available, for example on tape.

❖ **To recover the database:**

1 Make a copy of the database and log file.

2 Restore the database (*.db*) file, *not* the log file, from tape into a temporary directory.

3 In the temporary directory, start the database, applying the old logs using the `-a` command-line switch, applying the named transaction logs in the correct order.

4 Start the database using the current transaction log and the `-a` command-line switch, to apply the transactions and bring the database file up to date.

5 Start the database in your normal way. Any new activity will be appended to the current transaction log.

Example

Suppose you have a consolidated database file named *c:\dbdir\cons.db*. The transaction log file *c:\dbdir\cons.log* is mirrored to *d:\mirdir\cons.mlg*.

Assume that you perform full backups weekly, and you perform incremental backups daily using the following command:

```
dbbackup -c "uid=dba;pwd=sql" -r -t E:\BACKDIR
```

This command backs up the transaction log *cons.log* to the directory *e:\backdir*. The transaction log file is then renamed to *dateNN.log*, where *date* is the current date and *NN* is the next number in sequence, and a new transaction log is started. The directory *e:\backdir* is then backed up using a third-party utility.

In this scenario you would be running the Message Agent with the optional directory to point to the renamed transaction log files. The Message Agent command line would be

```
dbremote -c "uid=dba;pwd=sql" C:\DBDIR
```

On the third day following the weekly backup the database file gets corrupted because of a bad disk block.

❖ **To recover from media failure on the C: drive:**

1. Backup the mirrored transaction log *d:\mirdir\cons.mlg*.
2. Create a temporary directory to perform the recovery in. We will call it *c:\recover*.
3. Restore the most recent backup of the database file, *cons.db* to *c:\recover\cons.db*.
4. Apply the renamed transaction logs in order, as follows

   ```
   dbeng6 -a C:\DBDIR\date00.LOG C:\RECOVER\CONS.DB
   dbeng6 -a C:\DBDIR\date01.LOG C:\RECOVER\CONS.DB
   ```

5. Copy the current transaction log, *c:\dbdir\cons.log* to the recovery directory, giving *c:\recover\cons.log*.
6. Start the database using the following command:

   ```
   dbeng6 C:\RECOVER\CONS.DB
   ```

7. Shutdown the database server.
8. Backup the recovered database and transaction log from *c:\recover*.
9. Copy the files from *c:\recover* to the appropriate production directories.
   - Copy *c:\recover\cons.db* to *c:\dbdir\cons.db*.
   - Copy *c:\recover\cons.log* to *c:\dbdir\cons.log*, and to *d:\mirdir\cons.mlg*.
10. Restart your system as normal.

Backup procedures at remote databases

Backup procedures are not as crucial at remote databases as at the consolidated database. You may choose to rely on replication to the consolidated database as a data backup method. In the event of a media failure, the remote database would have to be re-extracted from the consolidated database, and any operations that have not been replicated would be lost. (You could use the log translation utility to attempt to recover lost operations.)

Even if you do choose to rely on replication to protect remote database data, backups still need to be done periodically at remote databases to prevent the transaction log from growing too large. You should use the same option (rename and restart the log) as at the consolidated database, running the Message Agent so that it has access to the renamed log files. If you set the DELETE_OLD_LOGS option to ON at the remote database, the old log files will be deleted automatically by the Message Agent when they are no longer needed.

Automatic transaction log renaming

You can use the `-x` Message Agent command-line switch to eliminate the need to rename the transaction log on the remote computer when the database server is shut down. The `-x` option renames transaction log after it has been scanned for outgoing messages.

Upgrading consolidated databases

This section describes issues in upgrading a consolidated database in a SQL Remote environment. The same considerations apply to Adaptive Server Anywhere databases that are primary sites in a Sybase Replication Server installation.

Installing new software does not always make new features available. In many cases, new features require the Upgrade utility to be run on databases. The Upgrade utility adds any information to the system catalog required for new features to be available. When you run the Upgrade utility, it tells you to archive the transaction log. The reason for this is that a new transaction log is created by the Upgrade utility, with a new file format.

When using SQL Remote or Replication Server, the transaction log must be kept for the Message Agent and the Replication Agent, respectively. After running the Upgrade utility, you should shut down the engine, rename the log, and leave it for the Message Agent to delete. The log should also be archived for backup purposes.

For information on the Upgrade utility, see the *Adaptive Server Anywhere User's Guide*.

Unloading and reloading a consolidated database

If a database is participating in replication, particular care needs to be taken if you wish to unload and reload the databases.

Replication is based on the transaction log. When a database is unloaded and reloaded, the old transaction log is no longer available. For this reason, good backup practices are especially important when participating in replication.

❖ **To unload and reload a consolidated database:**

1. Shut down the existing database.
2. Run the *dbtran* utility to display the starting offset and current log position. Note these values for later use.

   The following command lists the starting offset and current log offset for the database *asademo.db*:

   ```
   dbtran consol.log
   ```
3. Rename the current transaction log file so that it is not modified during the unload process.
4. Unload the database.
5. Reload the database.
6. Erase the current transaction log file.
7. Use dblog with the values noted in step 2 to set the new transaction log starting offset:

   ```
   dblog -z 137829 consol.db
   ```
8. Place the original log file in the same directory as the new log file using a different name.
9. When you run the Message Agent, provide it with the location of the renamed log file on its command line.

Using passthrough mode

The publisher of the consolidated database can directly intervene at remote sites using a passthrough mode, which enables standard SQL statements to be passed through to a remote site. By default, passthrough mode statements are executed at the local (consolidated) database as well, but an optional keyword prevents the statements from being executed locally.

> ***Caution***
> *Always test your passthrough operations on a test database with a remote database subscribed. Never run untested passthrough scripts against a production database.*

Starting and stopping passthrough

Passthrough mode is started and stopped using the PASSTHROUGH statement. Any statement entered between the starting PASSTHROUGH statement and the PASSTHROUGH STOP statement which terminates passthrough mode is checked for syntax errors, executed at the current database, and also passed to the identified subscriber and executed at the subscriber database. We can call the statements between a starting and stopping passthrough statement a **passthrough session**.

The following statement starts a passthrough session which passes the statements to a list of two named subscribers, without being executed at the local database:

```
PASSTHROUGH ONLY
FOR userid_1, userid_2;
```

Directing passthrough statements

The following statement starts a passthrough session which passes the statements to all subscribers to the specified publication:

```
PASSTHROUGH ONLY
FOR SUBSCRIPTION TO [owner].pubname [ ( string ) ] ;
```

Passthrough mode is additive. In the following example, **statement_1** is sent to **user_1**, and **statement_2** is sent to both **user_1** and **user_2**.

```
PASSTHROUGH ONLY FOR user_1 ;
statement_1 ;
PASSTHROUGH ONLY FOR user_2 ;
statement_2 ;
```

The following statement terminates a passthrough session:

```
PASSTHROUGH STOP ;
```

PASSTHROUGH STOP terminates passthrough mode for all remote users.

Order of application of passthrough statements

Passthrough statements are replicated in sequence with normal replication messages, in the order in which the statements are recorded in the log.

Passthrough is commonly used to send data definition language statements. In this case, replicated DML statements use the *before* schema before the passthrough and the *after* schema following the passthrough.

Notes on using passthrough mode

- You should always test your passthrough operations on a test database with a remote database subscribed. You should never run untested passthrough scripts against a production database.
- You should always qualify object names with the owner name. PASSTHROUGH statements are not executed at remote databases from the same user ID. Consequently, object names without the owner name qualifier may not be resolved correctly.

Uses and limitations of passthrough mode

Passthrough mode is a powerful tool, and should be used with care. Some statements, especially data definition statements, could cause a running SQL Remote setup to come tumbling down. SQL Remote relies on each database in a setup having the same objects: if a table is altered at some sites but not at others, attempts to replicate data changes will fail.

Also, it is important to remember that in the default setting passthrough mode also executes statements at the local database. To send statements to a remote database without executing them locally you must supply the ONLY keyword. The following set of statements drops a table not only at a remote database, but also at the consolidated database.

```
-- Drop a table at the remote database
-- and at the local database
PASSTHROUGH TO Joe_Remote ;
DROP TABLE CrucialData ;
PASSTHROUGH STOP ;
```

The syntax to drop a table at the remote database only is as follows:

```
-- Drop a table at the remote database only
PASSTHROUGH ONLY TO Joe_Remote ;
DROP TABLE CrucialData ;
PASSTHROUGH STOP ;
```

The following are tasks that can be carried out on a running SQL Remote setup:

- Add new users.
- Resynchronize users.

- Drop users from the setup.
- Change the address, message type, or frequency for a remote user.
- Add a column to a table.

Many other schema changes are likely to cause serious problems if executed on a running SQL Remote setup.

Passthrough works on only one level of a hierarchy

In a multi-tier SQL Remote installation, it becomes important that passthrough statements work on the level of databases immediately beneath the current level. In a multi-tier installation, passthrough statements must be entered at each consolidated database, for the level beneath it.

Operations not replicated in passthrough mode

There are special considerations for some statements in passthrough mode.

Calling procedures

When a stored procedure is called in passthrough mode using a CALL or EXEC statement, the CALL statement itself is replicated and none of the statements inside the procedure are replicated. It is assumed that the procedure on the replicate side has the correct effect.

Control of flow statements and cursor operations

Control-flow statements such as IF and LOOP, as well as any cursor operations, are not replicated in passthrough mode. Any statements within the loop or control structure *are* replicated.

Operations on cursors are not replicated. Inserting rows through a cursor, updating rows in a cursor, or deleting rows through a cursor are not replicated in passthrough mode.

Static embedded SQL SET OPTION statements are not replicated. The following statement is not replicated in passthrough mode:

```
EXEC SQL SET OPTION . . .
```

However, the following dynamic SQL statement is replicated:

```
EXEC SQL EXECUTE IMMEDIATE "SET OPTION . . . "
```

Batches

Batch statements (a group of statements surrounded with a BEGIN and END) are not replicated in passthrough mode. You receive an error message if you try to use batch statements in passthrough mode.