

CHAPTER 6

Principles of SQL Remote Design

About this chapter This chapter describes general issues and principles for designing a SQL Remote installation.

↪ For system-specific details, see the chapters "SQL Remote Design for Adaptive Server Enterprise" on page 159 and "SQL Remote Design for Adaptive Server Anywhere" on page 113.

Contents

Topic	Page
Design overview	96
How statements are replicated	100
Who gets what?	107
Replication errors and conflicts	109

Design overview

This chapter describes general publication design issues that you must address when designing a SQL Remote installation. It also describes how SQL Remote replicates data. SQL Remote is software for performing a complex task.

Design at the consolidated database


Like all SQL Remote administrative tasks, design is carried out by a database administrator or system administrator at the consolidated database.

The Adaptive Server Enterprise System Administrator or database administrator should perform all SQL Remote configuration tasks.

Ensuring compatible databases

You should ensure that all databases participating in a SQL Remote installation are compatible in terms of sort orders, character sets, and database option settings.

If your installation includes both Adaptive Server Enterprise and Adaptive Server Anywhere databases, you should ensure your Adaptive Server Anywhere databases are created in an Adaptive Server Enterprise-compatible fashion.

 For a full description of how to create Enterprise-compatible Adaptive Server Anywhere databases, see "Creating a Transact-SQL-compatible database", in the chapter "Using Transact-SQL with Adaptive Server Anywhere", in the *Adaptive Server Anywhere User's Guide*. This section provides a brief description only.

❖ To create an Enterprise-compatible Adaptive Server Anywhere database using Sybase Central:

- ◆ The Create Database wizard provides a button that sets each of the available choices to emulate Adaptive Server Enterprise. This is the simplest way to create a Transact-SQL-compatible database.

❖ To create an Enterprise-compatible Adaptive Server Anywhere database from the command line:

- 1 **Ensure trailing blanks are ignored** You can do this using the *dbinit -b* command-line switch.

- 2 **Ensure the dbo user ID is set** If you have a database that already has a user ID named **dbo**, then you can transfer the ownership of the Adaptive Server Anywhere Transact-SQL system views to another user ID. You can do this using the `dbinit -g` command-line switch.
- 3 **Remove historical system views** You can do this with the `dbinit -k` command-line switch.
- 4 **Make the database case sensitive** You can do this with the `dbinit -c` command-line switch.

The following command creates a case-sensitive database named `test.db` in the current directory, using the current **dbo** user, ignoring trailing blanks, and removing historical system views:

```
dbinit -b -c -k test.db
```

Using compatible sort orders and character sets

The SQL Remote Message Agent does not perform any character set conversions.

Character sets in Adaptive Server Anywhere installations

For an Adaptive Server Anywhere installation, the character set and collation used by the consolidated database must be the same as the remote databases. For information about supported character sets, see "Database Collations and International Languages" on page 289 of the book *Adaptive Server Anywhere User's Guide*.

Character sets in Adaptive Server Enterprise installations

The Open Client/Open Server libraries perform character set conversions between SSREMOTE and Adaptive Server Enterprise whenever the LOCALES.DAT character set is different from the Adaptive Server Enterprise character set. Both character sets must be installed on the Adaptive Server Enterprise server and conversion must be supported.

Character sets in mixed installations

The `locales.dat` settings (which are used by all Open Client applications) must match the remote Adaptive Server Anywhere settings.

The following table provides recommended matches between Adaptive Server Enterprise and Adaptive Server Anywhere character sets. The matches are not all complete.

Adaptive Server Anywhere collation name	Open Client / Open Server name	Open Client / Open Server case-sensitive sort order	Open Client / Open Server case-insensitive sort order
default	cp850	dictionary_cp850	nocase_cp850
850	cp850	dictionary_cp850	nocase_cp850
437	cp437	dictionary_cp437	nocase_cp437
852	cp852	bin_cp852	bin_cp852
860	cp860	bin_cp860	bin_cp860
437LATIN1	cp437	dictionary_cp437	nocase_cp437
437ESP	cp437	espdict_cp437	espnocs_cp437
437SVE	cp437	bin_cp437	bin_cp437
819CYR	iso_1	bin_iso_1	bin_iso_1
819DAN	iso_1	bin_iso_1	bin_iso_1
819ELL	iso_1	bin_iso_1	bin_iso_1
819ESP	iso_1	espdict_iso_1	espnocs_iso_1
819ISL	iso_1	bin_iso_1	bin_iso_1
819LATIN1	iso_1	dictionary_iso_1	nocase_iso_1
819LATIN2	iso_1	bin_iso_1	bin_iso_1
819NOR	iso_1	bin_iso_1	bin_iso_1
819RUS	iso_1	bin_iso_1	bin_iso_1
819SVE	iso_1	bin_iso_1	bin_iso_1
819TRK	iso_1	bin_iso_1	bin_iso_1
850CYR	cp850	bin_cp850	bin_cp850
850DAN	cp850	scandict_cp850	scannocp_cp850
850ELL	cp850	bin_cp850	bin_cp850
850ESP	cp850	espdict_cp850	espnocs_cp850
850ISL	cp850	scandict_cp850	scannocp_cp850
850LATIN1	cp850	dictionary_cp850	nocase_cp850
850LATIN2	cp850	bin_cp850	bin_cp850
850NOR	cp850	scandict_cp850	scannocp_cp850

Adaptive Server Anywhere collation name	Open Client / Open Server name	Open Client / Open Server case-sensitive sort order	Open Client / Open Server case-insensitive sort order
			0
850RUS	cp850	bin_cp850	bin_cp850
850SVE	cp850	scandict_cp850	scannocp_cp850
850TRK	cp850	bin_cp850	bin_cp850
852LATIN2	cp852	bin_cp852	bin_cp852
852CYR	cp852	bin_cp852	bin_cp852
855CYR	cp855	cyrdict_cp855	cynocs_cp855
857TRK	cp857	bin_cp857	bin_cp857
860LATIN1	cp860	bin_cp860	bin_cp860
866RUS	cp866	rusdict_cp866	rusnocs_cp866
869ELL	cp869	bin_cp869	bin_cp869
SJIS	sjis	bin_sjis	bin_sjis
SJIS2	sjis	bin_sjis	bin_sjis
EUC_JAPAN	eucjis	bin_eucjis	bin_eucjis
EUC_CHINA	eucgb	bin_eucgb	bin_eucgb
EUC_TAIWAN	eucb5	bin_big5	bin_big5
EUC_KOREA	eucksc	bin_eucksc	bin_eucksc
UTF8	utf8	bin_utf8	bin_utf8

How statements are replicated

	<p>SQL Remote replication is based on the transaction log, enabling it to replicate only changes to data, rather than all data, in each update. When we say that SQL Remote replicates data, we really mean that <i>SQL Remote replicates SQL statements that modify data</i>.</p>
Only committed transactions are replicated	<p>SQL Remote replicates only statements in committed transactions, to ensure proper transaction atomicity throughout the replication setup and maintain a consistency among the databases involved in the replication, albeit with some time lag while the data is replicated.</p>
Primary keys	<p>When an UPDATE or a DELETE is replicated, SQL Remote uses the primary key columns to uniquely identify the row being updated or deleted. All tables being replicated should have a declared primary key or uniqueness constraint. A unique index is not sufficient. The columns of the primary key are used in the WHERE clause of replicated updates and deletes. If a table has no primary key, the WHERE clause refers to all columns in the table.</p>
An UPDATE is not always an UPDATE	<p>When a simple INSERT statement is entered at one database, it is sent to other databases in the SQL Remote setup as an INSERT statement. However, not all statements are replicated exactly as they are entered by the client application. This section describes how SQL Remote replicates SQL statements. It is important to understand this material if you are to design a robust SQL Remote installation.</p> <p>The Message Agent is the component that carries out the replication of statements.</p>

Replication of inserts and deletes

INSERT and DELETE statements are the simplest replication case. SQL Remote takes each INSERT or DELETE operation from the transaction log, and sends it to all sites that subscribe to the row being inserted or deleted.

If only a subset of the columns in the table is subscribed to, the INSERT statements sent to subscribers contains only those columns.

The Message Agent ensures that statements are not replicated to the user that initially entered them.

Replication of updates

UPDATE statements are not replicated exactly as the client application enters them. This section describes two ways in which the replicated UPDATE statement may differ from the entered UPDATE statement.

UPDATE statements replicated as INSERTS or DELETES

If an UPDATE statement has the effect of removing a row from a given remote user's subscription, it is sent to that user as a DELETE statement. If an UPDATE statement has the effect of adding a row to a given remote user's subscription, it is sent to that user as an INSERT statement.

The figure illustrates a publication, where each subscriber subscribes by their name:

Consolidated			Ann		Marc	
ID	Rep	Dept	ID	Rep	ID	Rep
1	Ann	101	1	Ann	2	Marc
2	Marc	101			3	Marc
3	Marc	101				

Consolidated			Ann		Marc	
ID	Rep	Dept	ID	Rep	ID	Rep
1	Ann	101	1	Ann	2	Marc
2	Marc	101	3	Ann	3	Marc
3	Ann	101				

An UPDATE that changes the **Rep** value of a row from Marc to Ann is replicated to Marc as a DELETE statement, and to Ann as an INSERT statement.

This reassignment of rows among subscribers is sometimes called **territory realignment**, because it is a common feature of sales force automation applications, where customers are periodically reassigned among representatives.

UPDATE conflict detection

An UPDATE statement changes the value of one or more rows from some existing value to a new value. The rows altered depend on the WHERE clause of the UPDATE statement.

When SQL Remote replicates an UPDATE statement, it does so as a set of single-row updates. These single-row statements can fail for one of the following reasons:

- ◆ **The row to be updated does not exist** Each row is identified by its primary key values, and if a primary key has been altered by some other user, the row to be updated is not found.

In this case, the UPDATE does not update anything.

- ◆ **The row to be updated differs in one or more of its columns** If one of the values expected to be present has been changed by some other user, an **update conflict** occurs.

At remote databases, the update takes place regardless of the values in the row.

At the consolidated database, SQL Remote allows **conflict resolution** operations to take place. Conflict resolution operations are held in a trigger or stored procedure, and run automatically when a conflict is detected.

In Adaptive Server Anywhere, the conflict resolution trigger runs before the update, and the update proceeds when the trigger is finished. In Adaptive Server Enterprise, the conflict resolution procedure runs after the update has been applied.

- ◆ **A table without a primary key or uniqueness constraint refers to all columns in the WHERE clause of replicated updates** When two users update the same row, replicated updates will not update anything and databases will become inconsistent. All replicated tables should have a primary key or uniqueness constraint and the columns in the constraint should never be updated.

Replication of procedures

Any replication system is faced with a choice between two options when replicating a stored procedure call:

- ◆ **Replicate the procedure call** A corresponding procedure is executed at the replicate site, or
- ◆ **Replicate the procedure actions** The individual actions (INSERTs, UPDATES, DELETES and so on) of the procedure are replicated.

SQL Remote replicates procedures by replicating the actions of a procedure. The procedure call is not replicated.

Replication of triggers

Trigger replication from Adaptive Server Enterprise	<p>Trigger replication in SQL Remote is different for the Adaptive Server Enterprise Message Agent and the Adaptive Server Anywhere Message Agent.</p> <p>From Adaptive Server Enterprise, trigger actions are replicated. For this reason, care must be taken in the remote Adaptive Server Anywhere databases to be sure that triggers are not fired when operations are being applied by the Message Agent, or they are written so that the replicated trigger actions from the Adaptive Server Enterprise server do not cause a problem.</p>
Trigger replication from Adaptive Server Anywhere	<p>The FIRE_TRIGGERS Adaptive Server Anywhere database option prevents triggers from being fired. You can set this option for the user ID used by the Message Agent, but be careful to not use this user ID for other purposes. Alternatively, you can use CURRENT_REMOTE_USER in your triggers make some trigger code not execute when it is NULL when operations are being applied by the Message Agent.</p> <p>By default, the Message Agent for Adaptive Server Anywhere does not replicate actions performed by triggers; it is assumed that the trigger is defined remotely. This avoids permissions issues and the possibility of each action occurring twice. There are some exceptions to this rule:</p> <ul style="list-style-type: none"> ◆ Conflict resolution trigger actions The actions carried out by conflict resolution, or RESOLVE_UPDATE, triggers <i>are</i> replicated from a consolidated database to all remote databases, including the one that sent the message causing the conflict. ◆ Replication of BEFORE triggers Some BEFORE triggers can produce undesirable results when using SQL Remote, and so BEFORE trigger actions that modify the row being updated <i>are</i> replicated, before UPDATE actions. <p>You must be aware of this behavior when designing your installation. For example, a BEFORE_UPDATE that bumps a counter column in the row to keep track of the number of times a row is updated would double count if replicated, as the BEFORE_UPDATE trigger will fire when the UPDATE is replicated. To prevent this problem, you must ensure that, at the subscriber database, the trigger is not present or does not carry out the replicated action. Also, a BEFORE_UPDATE that sets a column to the time of the last update will get the time the UPDATE is replicated as well.</p>
An option to replicate trigger actions	<p>The Adaptive Server Anywhere Message Agent has a command-line switch that causes it to replicate all trigger actions when sending messages. This is the <i>dbremote -t</i> switch.</p>

If you use this switch, you must ensure that the trigger actions are not carried out twice at remote databases, once by the trigger being fired at the remote site, and once by the explicit application of the replicated actions from the consolidated database.

To ensure that trigger actions are not carried out twice, you can wrap an IF CURRENT REMOTE USER IS NULL ... END IF statement around the body of the triggers or you can set the Adaptive Server Anywhere Fire_triggers option to OFF for the Message Agent user ID.

Replication of data definition statements

Data definition statements (CREATE, ALTER, DROP, and others that modify database objects) are not replicated by SQL Remote unless they are entered while in passthrough mode.

↪ For information about passthrough mode for Adaptive Server Anywhere, see "Using passthrough mode" on page 273.

Replication of blobs

Blobs are LONG VARCHAR, LONG BINARY, TEXT, and IMAGE data types: values that are longer than 256 characters.

Adaptive Server
Anywhere
replication

SQL Remote includes a special method for replicating blobs between Adaptive Server Anywhere databases.

The Message Agent uses a variable in place of the value in the INSERT or UPDATE statement that is being replicated. The value of the variable is built up by a sequence of statements of the form

```
SET vble = vble || 'more_stuff'
```

This makes the size of the SQL statements involving long values smaller, so that they fit within a single message. The SET statements are separate SQL statements, so that the blob is effectively split over several SQL Remote messages.

Adaptive Server
Enterprise
replication

Some blobs can be replicated in SQL Remote installations including an Adaptive Server Enterprise, but there are limitations on the size of object that can be replicated. The objects being replicated must fit into half the maximum size of a single message.

❖ **To replicate blobs in a SQL Remote setup with Adaptive Server Enterprise:**

- 1 Ensure that all Message Agents in the system (both *dbremote* and *ssremote*) are running with a maximum message size greater than twice the size of the maximum blob size. You can configure the maximum message size using the `-l` command-line option.

If the maximum blob size is 100 Kb, run the Message Agents with **-l 220k**.

☞ For information on Message Agent command lines, see "The Message Agent" on page 306.

- 2 Set the `BLOB_THRESHOLD` database option to a value larger than the largest blob.

For example, with a maximum blob size of 100Kb, you could set `BLOB_THRESHOLD` to 110k. If you have SQL Anywhere 5.5.04 or earlier in your system, it will complain about `BLOB_THRESHOLD` being an unknown option: you can ignore this error.

☞ For information about setting options, see "SQL Remote options" on page 323.

Sybase Open Client CTLIB applications that manipulate the `CS_IODESC` structure must not set the `log_on_update` member to `FALSE`.

The Message Agent for Adaptive Server Anywhere may be slow when applying the messages with large blobs.

Using the `Verify_threshold` option to minimize message size

The `Verify_threshold` database option can prevent long values from being verified (in the `VERIFY` clause of a replicated `UPDATE`). The default value for the option is 1000. If the data type of a column is longer than the threshold, old values for the column are not verified when an `UPDATE` is replicated. This keeps the size of SQL Remote messages down, but has the disadvantage that conflicting updates of long values are not detected.

There is a technique allowing detection of conflicts when `Verify_threshold` is being used to reduce the size of messages. Whenever a "blob" is updated, a `last_modified` column in the same table should also be updated. Conflicts can then be detected because the old value of the `last_modified` column is verified.

Using a work table to avoid redundant updates

Repeated updates to a blob should be done in a "work" table, and the final version should be assigned to the replicated table. For example, if a document in progress is updated 20 times throughout the day and the Message Agent is run once at the end of the day, all 20 updates are replicated. If the document is 200Kb in length, this causes 4Mb of messages to be sent.

Controlling
replication of blobs

The better solution is to have a **document_in_progress** table. When the user is done revising a document, the application moves it from the **document_in_progress** table to the replicated table. The results in a single update (200Kb of messages).

The Adaptive Server Anywhere BLOB_THRESHOLD option allows further control over the replication of long values. Any value longer than the BLOB_THRESHOLD option is replicated as a blob. That is, it is broken into pieces and replicated in chunks, before being reconstituted by using a SQL variable and concatenating the pieces at the recipient site.

By setting BLOB_THRESHOLD to a high value in remote Adaptive Server Anywhere databases, blobs are not broken into pieces, and operations can be applied to Adaptive Server Enterprise by the Message Agent. Each SQL statement must fit within a message, so this only allows replication of small blobs.

Who gets what?

Each time a row in a table is inserted, deleted, or updated, a message has to be sent to those subscribed to the row. In addition, an update may cause the subscription expression to change, so that the statement is sent to some subscribers as a delete, some as an update, and some as an insert.

☞ For details of what statements get sent to which subscribers, see "How statements are replicated" on page 100. For details on subscriptions, see the following two chapters.

This section describes how SQL Remote sends the right operations to the right recipients.

The task of determining who gets what is divided between the database server and the Message Agent. The engine handles those aspects that are to do with publications, while the Message Agent handles aspects to do with subscriptions.

Adaptive Server Anywhere actions

Adaptive Server Anywhere evaluates the subscription expression for each update made to a table that is part of a publication. It adds the value of the expression to the log, both before and after the update.

Not the subscriber list

Adaptive Server Enterprise does not evaluate or enter into the log a list of subscribers. The subscription expression (a property of the publication) is evaluated and entered. All handling of subscribers is left to the Message Agent.

For a table that is part of more than one publication, the subscription expression is evaluated before and after the update for each publication.

The addition of information to the log can affect performance in the following cases:

- ◆ **Expensive expressions** When a subscription expression is expensive to evaluate, it can affect performance.
- ◆ **Many publications** When a table belongs to many publications, many expressions must be evaluated. In contrast, the number of *subscriptions* is irrelevant.
- ◆ **Many-valued expressions** Some expressions are many-valued. This can lead to much additional information in the transaction log, with a corresponding effect on performance.

Adaptive Server Enterprise actions

In a SQL Remote for Adaptive Server Enterprise publication, the subscription expression must be a column. The subscription column contains either a single value or a comma-separated list of values.

Not the subscriber list

Adaptive Server Enterprise does not enter into the log a list of subscribers. The column value is entered. All handling of subscribers is left to the Message Agent.

When a table is marked for replication using `sp_add_remote_table` (which calls `sp_setreplicate`), Adaptive Server Enterprise places an entire before image of the row in the transaction log for deletes, and entire after image for inserts, and both images for updates. This means that the before and after values of the subscription column are available.

Message Agent actions

The Message Agent reads the evaluated subscription expressions or subscription column entries from the transaction log, and matches the before and after values against the subscription value for each subscriber to the publication. In this way, the Message Agent can send the correct operations to each subscriber.

While large numbers of subscribers do not have any impact on server performance, they can impact Message Agent performance. Both the work in matching subscription values against large numbers of subscription values, and the work in sending the messages, can be demanding.

Replication errors and conflicts

SQL Remote is designed to allow databases to be updated at many different sites. Careful design is required to avoid replication errors, especially if the database has a complicated structure. This section describes the kinds of errors and conflict that can occur in a replication setup; subsequent sections describe how you can design your publications to avoid errors and manage conflicts.

Delivery errors not discussed here

This section does not discuss issues related to message delivery failures. For information on delivery errors and how they are handled, see "The message tracking system" on page 252

Replication errors

Replication errors fall into the following categories:

- ◆ **Duplicate primary key errors** Two users INSERT a row using the same primary key values, or one user updates a primary key and a second user inserts a primary key of the new value. The second operation to reach a given database in the replication system fails because it would produce a duplicate primary key.
- ◆ **Row not found errors** A user DELETES a row (that is, the row with a given primary key value). A second user UPDATES or DELETES the same row at another site.

In this case, the second statement fails, as the row is not found.

- ◆ **Referential integrity errors** If a column containing a foreign key is included in a publication, but the associated primary key is not included, the extraction utility leaves the foreign key definition out of the remote database so that INSERTS at the remote database will not fail.

This can be solved by including proper defaults into the table definitions.


Also, referential integrity errors can occur when a primary table has a SUBSCRIBE BY expression and the associated foreign table does not: rows from the foreign table may be replicated, but the rows from the primary table may be excluded from the publication.

Replication conflicts

Replication conflicts are different from errors. Properly handled, conflicts are not a problem in SQL Remote.

- ◆ **Conflicts** A user updates a row. A second user updates the same row at another site. The second user's operation succeeds, and SQL Remote allows a trigger to be fired (Adaptive Server Anywhere) or a procedure to be called (Adaptive Server Enterprise) to resolve these conflicts in a way that makes sense for the data being changed.

Conflicts will occur in many installations. SQL Remote allows appropriate resolution of conflicts as part of the regular operation of a SQL Remote setup, using triggers and procedures.

 For information about how SQL Remote handles conflicts as they occur, see the following chapters.

Tracking SQL errors

SQL errors in replication must be designed out of your setup. SQL Remote includes an option to help you track errors in SQL statements, but this option is not intended to resolve such errors.

By setting the **Replication_error** option, you can specify a stored procedure to be called by the Message Agent when a SQL error occurs. By default no procedure is called.

❖ To set the **Replication_error** option in Adaptive Server Anywhere:

- ◆ Issue the following statement:

```
SET OPTION
remote-user.Replication_error
= 'procedure-name'
```

where *remote-user* is the user ID on the Message Agent command line, and *procedure-name* is the procedure called when a SQL error is detected.

❖ To set the **Replication_error** option in Adaptive Server Enterprise:

- ◆ Issue the following statement:

```
exec sp_remote_option Replication_error, procedure-
name
go
```


where *procedure-name* is the procedure called when a SQL error is detected.

Replication error
procedure
requirements

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

