Contents

About this Manual................................................................. ix
  Documentation conventions............................................ x
  The sample database....................................................... xiv

PART ONE

Introduction to SQL Remote .............................................. 1

1 Welcome to SQL Remote .................................................. 3
  About SQL Remote.......................................................... 4
  About this manual.......................................................... 7

2 SQL Remote Replication Concepts .................................... 9
  Introduction to data replication........................................ 10
  SQL Remote concepts.................................................... 13
  SQL Remote components............................................... 17
  Publications and subscriptions........................................ 20
  SQL Remote features..................................................... 22
  Some sample installations.............................................. 24

3 Setting Up SQL Remote .................................................. 29
  Setup overview............................................................ 30
  Preparing your Adaptive Server Enterprise server................. 31
  Uninstalling SQL Remote............................................... 35
  Upgrading SQL Remote for Adaptive Server Enterprise........... 36

4 A Tutorial for Adaptive Server Anywhere Users................. 37
  Tutorial overview........................................................ 38
  A tutorial using Sybase Central........................................ 41
  Setting up a consolidated database................................... 44
  Set up the remote database in Sybase Central....................... 49
  A tutorial using Interactive SQL and DBXTRACT..................... 51
  Set up the consolidated database................................. 53
Set up the remote database ........................................ 57
Start replicating data ............................................... 59
A sample publication ............................................. 63

5

A Tutorial for Adaptive Server Enterprise Users .......... 65
Introduction ......................................................... 66
Setting up SQL Remote using Sybase Central ............. 69
Setting up the consolidated database ...................... 72
Set up the remote database in Sybase Central .......... 77
A tutorial using isolate and ssxtract ....................... 79
Setting up the consolidated database .................... 82
Start replicating data ........................................... 88

PART TWO

Replication Design for SQL Remote ....................... 93

6

Principles of SQL Remote Design ......................... 95
Design overview .................................................. 96
How statements are replicated .............................. 100
Who gets what? ................................................. 107
Replication errors and conflicts ............................ 109

7

SQL Remote Design for Adaptive Server Anywhere ... 113
Design overview .................................................. 114
Creating publications ......................................... 115
Publication design for Adaptive Server Anywhere .... 122
Partitioning tables that do not contain the
subscription expression ........................................ 125
Sharing rows among several subscriptions ............ 133
Managing conflicts ............................................. 142
Ensuring unique primary keys ............................. 152
Creating subscriptions ....................................... 157

8

SQL Remote Design for Adaptive Server Enterprise .. 159
Design overview .................................................. 160
Creating publications ......................................... 161
Publication design for Adaptive Server Enterprise .... 166
Partitioning tables that do not contain the
subscription column ............................................. 168
Sharing rows among several subscriptions ............ 175
Managing conflicts .......................................................... 182
Ensuring unique primary keys .......................................... 192
Creating subscriptions ..................................................... 198

PART THREE

SQL Remote Administration .............................................. 201

9

Deploying and Synchronizing Databases ......................... 203
Deployment overview ..................................................... 204
Test before deployment .................................................. 205
Synchronizing databases ............................................... 207
Using the extraction utility ............................................. 209
Synchronizing data over a message system .................... 216

10

SQL Remote Administration ........................................... 217
Management overview ................................................... 218
Managing SQL Remote permissions ................................. 219
Using message types .................................................... 228
Running the Message Agent ......................................... 239
Tuning Message Agent performance .............................. 243
Encoding and compressing messages ......................... 250
The message tracking system ...................................... 252

11

Administering SQL Remote for Adaptive Server Anywhere ................................................................. 255
Running the Message Agent ........................................... 256
Error reporting and handling ......................................... 258
Transaction log and backup management ..................... 262
Using passthrough mode ............................................. 273

12

Administering SQL Remote for Adaptive Server Enterprise ................................................................. 277
How the Message Agent for Adaptive Server Enterprise works ......................................................... 278
Running the Message Agent ......................................... 283
Error reporting and handling ......................................... 285
Adaptive Server Enterprise transaction log and backup management ................................................. 286
Making schema changes ............................................... 289
Using passthrough mode ............................................. 290
REMOTE RESET statement .............................................. 374
REVOKE CONSOLIDATE statement ................................ 375
REVOKE PUBLISH statement .......................................... 376
REVOKE REMOTE statement .......................................... 377
REVOKE REMOTE DBA statement .................................... 378
START SUBSCRIPTION statement ................................... 379
STOP SUBSCRIPTION statement ...................................... 380
SYNCHRONIZE SUBSCRIPTION statement ....................... 381
UPDATE statement .................................................... 382

Command Reference for Adaptive Server Enterprise. 385
sp_add_article procedure .............................................. 387
sp_add_article_col procedure ....................................... 388
sp_add_remote_table procedure ..................................... 389
sp_create_publication procedure .................................. 391
sp_drop_publication procedure ...................................... 392
sp_drop_remote_type procedure ...................................... 393
sp_drop_sql_remote procedure ....................................... 394
sp_grant_consolidate procedure .................................... 395
sp_grant_remote procedure ............................................ 397
sp_modify_article procedure ......................................... 399
sp_modify_remote_table procedure .................................. 401
sp_passthrough procedure ............................................ 403
sp_passthrough_piece procedure .................................... 404
sp_passthrough_stop procedure ...................................... 406
sp_passthrough_subscription procedure ......................... 407
sp_passthrough_user procedure ...................................... 408
sp_populate_sql_anywhere procedure ............................. 409
sp_publisher procedure ................................................ 410
sp_queue_clean procedure ............................................. 411
sp_queue_delete_old procedure ...................................... 412
sp_queue_drop procedure .............................................. 413
sp_queue_dump_database procedure ................................. 414
sp_queue_dump_transaction procedure ............................ 415
sp_queue_get_state procedure ....................................... 416
sp_queue_read procedure .............................................. 417
sp_queue_reset procedure ............................................. 418
sp_queue_set_confirm procedure .................................... 419
sp_queue_set_progress procedure .................................. 420
sp_queue_transaction procedure .................................... 421
sp_remote procedure .................................................. 422
sp_remote_option procedure ......................................... 423
sp_remote_type procedure ............................................ 424
sp_remove_article procedure ......................................... 425
sp_remove_article_col procedure ................................... 426
About this Manual

Subject
This manual describes SQL Remote Version 6, the message-based two-way replication system designed for mobile computing.

Audience
This manual is for users of Adaptive Server Anywhere and Adaptive Server Enterprise who wish to add SQL Remote replication to their information systems.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation conventions</td>
<td>x</td>
</tr>
<tr>
<td>The sample database</td>
<td>xiv</td>
</tr>
</tbody>
</table>
Documentation conventions

This section lists the typographic and graphical conventions used in this documentation.

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- **Keywords**  All SQL keywords are shown in UPPER CASE. However, SQL keywords are case insensitive, so you can enter keywords in any case you wish; SELECT is the same as Select which is the same as select.

- **Placeholders**  Items that must be replaced with appropriate identifiers or expressions are shown in *italics*.

- **Continuation**  Lines beginning with ... are a continuation of the statements from the previous line.

- **Repeating items**  Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots). One or more list elements are allowed. If more than one is specified, they must be separated by commas.

- **Optional portions**  Optional portions of a statement are enclosed by square brackets. For example, ...

  ```sql
  RELEASE SAVEPOINT [ savepoint-name ]
  ```

  ... indicates that the *savepoint-name* is optional. The square brackets should not be typed.

- **Options**  When none or only one of a list of items must be chosen, the items are separated by vertical bars and the list enclosed in square brackets.

  For example, ...

  ```sql
  [ ASC | DESC ]
  ```

  ... indicates that you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

- **Alternatives**  When precisely one of the options must be chosen, the alternatives are enclosed in curly braces. For example ...

  ```sql
  QUOTES { ON | OFF }
  ```

  ... indicates that exactly one of ON or OFF must be provided. The braces should not be typed.
Graphic icons

The following icons are used in this documentation:
<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Client Application Icon" /></td>
<td>A client application.</td>
</tr>
<tr>
<td><img src="image" alt="Database Server Icon" /></td>
<td>A database server, such as Sybase Adaptive Server Anywhere or Adaptive Server Enterprise.</td>
</tr>
<tr>
<td><img src="image" alt="Adaptive Server Anywhere Icon" /></td>
<td>An Adaptive Server Anywhere database server.</td>
</tr>
<tr>
<td><img src="image" alt="SQL Remote Message Agent Icon" /></td>
<td>A SQL Remote Message Agent</td>
</tr>
<tr>
<td><img src="image" alt="Sybase Replication Server Icon" /></td>
<td>A Sybase Replication Server.</td>
</tr>
</tbody>
</table>
## Icon | Meaning
--- | ---
![Icon](image) | A Replication Agent. A replication agent is required for a database to act as a primary data site in Sybase Replication Server installations.

## Installed files

The following terms are used throughout the manual:

- **Installation directory**  The directory into which you install Adaptive Server Anywhere.
- **Executable directory**  The executables and other files for each operating system are held in an executable subdirectory of the installation directory. This subdirectory has the following name:
  - **Windows NT and Windows 95**  win32
  - **UNIX**  bin
  - **Windows 3.x**  win
  - **Novell NetWare**  nlm
The sample database

There is a sample database included with Adaptive Server Anywhere. Many of the examples throughout the documentation use this sample database.

The sample database represents a small company. It contains internal information about the company (employees, departments, and financial data) as well as product information (products), sales information (sales orders, customers, and contacts), and financial information (fin_code, fin_data).

The following figure shows the tables in the sample database and how they are related to each other.

The sample database is held in a file named asademo.db, and is located in your Adaptive Server Anywhere installation directory.
This part describes the concepts, architecture, and features of SQL Remote. The material in this part refers to both SQL Remote for Adaptive Server Anywhere and SQL Remote for Adaptive Server Enterprise.
CHAPTER 1

Welcome to SQL Remote

About this chapter

This chapter introduces SQL Remote and the documentation.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About SQL Remote</td>
<td>4</td>
</tr>
<tr>
<td>About this manual</td>
<td>7</td>
</tr>
</tbody>
</table>
About SQL Remote

SQL Remote is a data-replication technology designed for two-way replication between a consolidated data server and large numbers of remote databases, typically including many mobile databases.

SQL Remote replication is message based, and requires no direct server-to-server connection. An occasional dial-up or e-mail link is sufficient.

Administration and resource requirements at the remote sites are minimal, and a typical time lag between the consolidated and remote databases is on the order of hours or days.

Sybase SQL Remote technology is provided in two forms:

- **SQL Remote for Adaptive Server Anywhere** Enables replication between a consolidated Adaptive Server Anywhere database and a large number of remote databases.
- **SQL Remote for Adaptive Server Enterprise** Enables replication between a consolidated Adaptive Server Enterprise database and a large number of remote databases.

<table>
<thead>
<tr>
<th>Adaptive Server Enterprise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sybase Adaptive Server Enterprise was previously known as Sybase SQL Server.</td>
</tr>
</tbody>
</table>

This book describes both of these technologies.

In a SQL Remote installation, you must have properly licensed SQL Remote software at each participating database.

🔗 For a detailed introduction to SQL Remote concepts and features, see "SQL Remote Replication Concepts" on page 9.

### Supported platforms and environments

<table>
<thead>
<tr>
<th>Supported Database Management Systems</th>
<th>SQL Remote supports both Sybase Adaptive Server Enterprise and Sybase Adaptive Server Anywhere database management systems. Not all operating systems and message systems are supported for both these DBMS's.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported message systems</td>
<td>SQL Remote exchanges data among databases using an underlying message system. SQL Remote supports the following message systems:</td>
</tr>
<tr>
<td></td>
<td><strong>File sharing</strong> A simple system requiring no extra software.</td>
</tr>
</tbody>
</table>

4
Supported operating systems

- **FTP** Internet file transfer protocol.
- **SMTP/POP** Internet e-mail protocol.
- **MAPI** Microsoft Messaging Application Programming Interface, used in Microsoft products and in cc:Mail release 8 and later.
- **VIM** Vendor Independent Messaging, used in Lotus Notes and in some versions of Lotus cc:Mail.

Not all systems are supported on all operating systems.

SQL Remote for Adaptive Server Enterprise is available for the following operating systems:

- **Windows NT** All message protocols.
- **Sun Microsystems Solaris/Sparc** File sharing only.
- **Hewlett-Packard HP-UX** File sharing only.
- **Silicon Graphics IRIX** File sharing only.

```markdown
SQL Remote Open Server on IRIX
A bug in IRIX that was fixed in the IRIX 10.0.3 release EBF 7658 is required for the SQL Remote Open Server. `ssqueue` systems involving SGI/IRIX should make sure this patch is installed.
```

- **IBM AIX** File sharing only.

SQL Remote for Adaptive Server Anywhere is available for the following operating systems:

- Windows 95
- Windows NT
- Windows 3.x
- Sun Microsystems Solaris/Sparc
- Hewlett Packard HP-UX
- Silicon Graphics IRIX
- Novell NetWare.
- IBM OS/2

Only the file sharing protocol is available on the UNIX and NetWare operating systems.
The mobile Intranet

SQL Remote and Sybase PowerDynamo enable the development of mobile intranets. PowerDynamo is a set of tools for building database-hosted Web sites, and these sites can be replicated using SQL Remote onto laptop computers for offline browsing using the PowerDynamo Personal Web Server.

There are some special considerations when replicating a Web site. For information about using SQL Remote with PowerDynamo, you should read the PowerDynamo User's Guide.
About this manual

This manual describes how to design, build, and maintain SQL Remote installations.

The manual includes the following parts.

- Introduction to SQL Remote  Replication concepts and features of SQL Remote.
- Replication Design for SQL Remote  Designing SQL Remote installations.
- SQL Remote Administration  Deploying SQL Remote databases and administering a running SQL Remote setup.
- Reference  SQL Remote commands, system tables, and other reference material.

Platforms and products covered by this manual

This documentation applies to all supported platforms of SQL Remote. Not all features are available on all platforms. The operating systems and features you are entitled to use depends on the product you have purchased and your license agreement.

Available formats  The contents of this manual are provided online and in print. In some cases, the print documentation must be purchased separately.

Online documentation most current
The online documentation contains the most current information about SQL Remote. It is more authoritative than the printed manual.

Product installation

This section describes installation of SQL Remote for Adaptive Server Enterprise. If you obtained SQL Remote as part of another product, consult the installation instructions for the product you purchased.

❖ To install the SQL Remote software for PC's:

1  Insert the CD-ROM into your CD-ROM drive.
2  If the installation program does not start automatically, start the setup application on the CD-ROM.
3 Follow the instructions in the installation program.

❖ To install the SQL Remote software for UNIX:
  ♦ Consult the instructions for your operating system in the Adaptive Server Anywhere Read Me First booklet.

赒 If you are using SQL Remote for Adaptive Server Enterprise, you must install SQL Remote into any database you wish to replicate. For information about installing SQL Remote into a database, see "Setting Up SQL Remote" on page 29.
CHAPTER 2

SQL Remote Replication Concepts

About this chapter
This chapter introduces the concepts, design goals, and features of SQL Remote.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to data replication</td>
<td>10</td>
</tr>
<tr>
<td>SQL Remote concepts</td>
<td>13</td>
</tr>
<tr>
<td>SQL Remote components</td>
<td>17</td>
</tr>
<tr>
<td>Publications and subscriptions</td>
<td>20</td>
</tr>
<tr>
<td>SQL Remote features</td>
<td>22</td>
</tr>
<tr>
<td>Some sample installations</td>
<td>24</td>
</tr>
</tbody>
</table>
Introduction to data replication

Data replication is the sharing of data among physically distinct databases. Changes made to shared data at any one database are replicated to the other databases in the replication setup. The SQL Remote data replication system enables replication of data among Adaptive Server Anywhere or Adaptive Server Enterprise databases.

Benefits of data replication

| Data availability | One of the key benefits of a data replication system is that data is made available locally, rather than through potentially expensive, less reliable, and slow connections to a single central database. Data is accessible locally even in the absence of any connection to a central server, so that you are not cut off from data in the event of a failure of a long-distance network connection. |
| Response time | Replication improves response times for data requests for two reasons. Requests are processed on a local server without accessing some wide area network, so that retrieval rates are faster. Also, local processing offloads work from a central database server so that competition for processor time is decreased. |

Challenges for replication technologies

| Transactional integrity | One of the challenges of any replication system is to ensure that each database retains transactional integrity at all times. Today's replication systems, such as Sybase Replication Server and SQL Remote, replicate portions of the transaction log in such a way that transactions are replicated atomically: either a whole transaction is replicated, or none of it is replicated. This ensures transactional integrity at each database in the setup. |
| Data consistency | Another challenge to replication systems is to maintain data consistency throughout the setup. SQL Remote maintains a loose consistency in the setup as a whole: that is, all changes are replicated to each site over time in a consistent manner, but because of the time lag different sites may have different copies of data at any instant. |
Sybase replication technologies

Sybase provides two replication technologies; SQL Remote and Replication Server.

- **SQL Remote** is designed for two-way replication involving a consolidated data server and large numbers of remote databases, typically including many mobile databases. Administration and resource requirements at the remote sites are minimal, and a typical time lag between the consolidated and remote databases is on the order of hours.

- **Replication Server** is designed for replication among relatively small numbers of data servers, with a typical time lag between primary data and replicate data of a few seconds, and generally with an administrator at each site.

Choosing a replication technology

SQL Remote is designed for replication installations with the following requirements:

- **Large numbers of databases** SQL Remote is designed to support potentially large number of remote databases. It can support thousands of remote databases in a single installation.

- **Occasionally connected** Remote databases that are occasionally connected or indirectly connected to the network on which the server is running.

- **High latency** High latency means a long lag time between data being entered at one database and being replicated to each database in the installation. With SQL Remote, replication messages are sent typically at periods of hours or days.

- **Low volume** As replication messages are delivered occasionally, high volume can lead to very large numbers of messages being sent. SQL Remote is best suited to systems with a relatively low volume of replicated data per database.

- **Homogeneous databases** SQL Remote supports Adaptive Server Enterprise and Adaptive Server Anywhere databases. Each database in the system must have a very similar schema.

Replication Server is designed for replication installations with the following requirements:

- **Small numbers of databases** Replication Server is designed to support replication among servers, with installations typically involving under one hundred servers.
Introduction to data replication

- **Continuously connected** Connections between primary sites and replicate sites may be over a wide area network, but Replication Server is designed for situations where there is a near-continuous connection path for data exchange among the servers in the installation.

- **Low latency** Low latency means a short lag time between data being entered at one database and being replicated to each database in the installation. With Replication Server, replication messages are sent typically within seconds of being entered at a primary site.

- **High volume** With near-continuous connections and high performance, Replication Server is designed for a high volume of replication messages.

- **Heterogeneous databases** Replication Server supports several leading DBMS's, and allows mapping of object names during replication, so that support for heterogeneous databases is provided.

**Sybase SQL Remote**

Sybase SQL Remote technology is provided in two forms:

- **SQL Remote for Adaptive Server Anywhere** Enables replication between a consolidated Adaptive Server Anywhere database and a large number of remote databases.

- **SQL Remote for Adaptive Server Enterprise** Enables replication between a consolidated Adaptive Server Enterprise database and a large number of remote databases.

This book describes both of these technologies.

In a SQL Remote installation, you must have properly licensed SQL Remote software at each participating database.
SQL Remote concepts

This section introduces concepts and terms used throughout the book.

Consolidated and remote databases

SQL Remote provides data replication between a **consolidated database** and a set of **remote databases**.

A **consolidated database** is a database that contains all the data to be replicated. A **remote database** is a database that may be running at the same site as the consolidated database or at a physically distant site.

The figure shows a schematic illustration of a small SQL Remote installation.

Remote users

A replication installation includes many copies of the information in a database. Each copy is a physically separate database on a separate computer. All remote copies must stay consistent with the consolidated database.

The entire replication setup may be considered a single **dispersed database**, with the master copy of all data being kept at the consolidated database.

Each remote site that submits replications to the consolidated database is considered to be a **remote user** of the consolidated database. In the case that a remote site is a multi-user server, the entire site is considered to be a single remote user of the consolidated database.

Hierarchical database configurations

SQL Remote supports **hierarchical configurations** of databases; it does not support peer-to-peer replication or other non-hierarchical configurations.
SQL Remote concepts

For any two databases in a hierarchical configuration, one is always above or below the other in the hierarchy.

For databases in a non-hierarchical configuration, there is not any well-defined notion of above or below.

In a SQL Remote installation, each database contains all or a subset of the data replicated by the database above it in the hierarchy.

Remote databases can contain tables that are not present at the consolidated database, as long as they are not involved in replication.

Two-way replication

SQL Remote replication is two-way: changes made at the consolidated database are replicated to remote databases, while changes made at remote databases are replicated to the consolidated database, and thence to other remote databases.
CHAPTER 2  SQL Remote Replication Concepts

Message-based replication

SQL Remote exchanges data between databases using messages. This allows replication between databases that have no direct connection: an occasional message-based connection such as e-mail or a periodic dial-up link is sufficient.

In message-based communications, each message carries its destination address and other control information, so that no direct connection is needed between applications exchanging information. For example, an e-mail message contains the destination address; there is no direct connection between the sending server and the recipient.

Just as session-based client/server applications rely on network communication protocol stacks, such as TCP/IP or Novell NetWare's IPX, so message-based applications rely on message services such as Internet Simple Mail Transfer Protocol (SMTP), Microsoft's Messaging API (MAPI), Lotus' Vendor Independent Messaging (VIM), or a simple shared file link.

Message services use store-and-forward methods to get each message to its destination: for example, e-mail systems store messages until the recipient opens their mail folder to read their mail, at which time the e-mail system forwards the message.

Building a replication system on top of a message system means that SQL Remote does not need to implement a store-and-forward system to get messages to their destination. Just as session-based client/server applications do not implement their own protocol stacks to pass information between client and server, so SQL Remote uses existing message systems to pass the messages.
Guaranteed delivery

Unlike some session-based communications, many message-based systems do not guarantee that messages reach their destination, or that messages are received in the same order they were sent.

SQL Remote incorporates a protocol to guarantee reception of replication updates in the correct order.
SQL Remote components

The following components are required for SQL Remote:

- **Data server**  An Adaptive Server Anywhere or Adaptive Server Enterprise database management system is required at each site to maintain the data.

- **Message Agent**  A SQL Remote Message Agent is required at the consolidated site and at each remote site to send and receive SQL Remote messages.

  The Message Agent connects to the data server by a client/server connection. It may run on the same machine as the data server or on a different machine.

- **Database extraction utility**  The extraction utility is used to prepare remote databases from a consolidated database, during development and testing, and also at deployment time.

- **Message system client software**  Each computer involved in a setup must have a message system client installed. SQL Remote does support a file-sharing "message system", which does not require client software in addition to the SMTP/POP, MAPI, and VIM message systems under some operating systems.

- **Client applications**  The applications that work with SQL Remote databases are standard client/server database applications.
The data server

The data server may be an Adaptive Server Enterprise or an Adaptive Server Anywhere server. At the remote site the data server is commonly an Adaptive Server Anywhere personal server, but can also be an Adaptive Server Enterprise or Adaptive Server Anywhere server.

Client applications

Client applications work with the data in the database. Client applications use one of the client/server interfaces supported by the data server:

- For Adaptive Server Anywhere, the client application may use ODBC, Embedded SQL, or Sybase Open Client to work with Adaptive Server Anywhere.
- For Adaptive Server Enterprise, the client application may use one of the Sybase Client Server interfaces, ODBC, or Embedded SQL.

Client applications do not have to know if they are using a consolidated or remote database. From the client application perspective, there is no difference.

The Message Agent

The SQL Remote Message Agent sends and receives replication messages. It is a client application that sends and receives messages from database to database. The Message Agent must be installed at both the consolidated and at the remote sites.

For Adaptive Server Anywhere, the Message Agent is a program called dbremote.exe on PC operating systems, and dbremote on UNIX.

For Adaptive Server Enterprise, the Message Agent is a program called ssremote.exe on PC operating systems, and ssremote on UNIX.
**Message system client**

If you are using a shared file message system, no message system client is needed.

If you are using an e-mail or other message system, you must have a message system for that client in order to send and receive messages.
Publications and subscriptions

The data that is replicated by SQL Remote is arranged in publications. Each database that shares information in a publication must have a subscription to the publication.

The publication is a database object describing data to be replicated. Remote users of the database who wish to receive a publication do so by subscribing to a publication.

A publication may include data from several database tables. Each table's contribution to a publication is called an article. Each article may consist of a whole table, or a subset of the rows and columns in a table.

A two-table publication

<table>
<thead>
<tr>
<th>Article 1: all of table A</th>
<th>+</th>
<th>Article 2: some rows and columns from table B</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ ✓ ✓ ✓ ✓</td>
<td>+</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>✓ ✓ ✓ ✓ ✓</td>
<td></td>
<td>✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>

Periodically, the changes made to each publication in a database are replicated to all subscribers to that publication. These replications are called publication updates.

Remote databases subscribe to publications on the consolidated database so that they can receive data from the consolidated database. To do this, a subscription is created at the consolidated database, identifying the subscriber by name and by the publication they are to receive.

SQL Remote always involves messages being sent two ways. The consolidated database sends messages containing publication updates to remote databases, and remote databases also send messages to the consolidated database.

Messages are always sent both ways
For example, if data in a publication at a consolidated database is updated, those updates are sent to the remote databases. And even if the data is never updated at the remote database, **confirmation messages** must still be sent back to the consolidated database, to keep track of the status of the replication.

**Both databases subscribe**

Messages must be sent both ways, so not only does a remote database subscribe to a publication created at the consolidated database, but the consolidated database must subscribe to a corresponding publication created at the remote database.

![Diagram showing publication and subscription between consolidated and remote databases](image)

When remote database users modify their own copies of the data, their changes are replicated to the consolidated database. When the messages containing the changes are applied at the consolidated database the changes become part of the consolidated database's publication, and are included in the next round of updates to all remote sites (except the one it came from). In this way, replication from remote site to remote site takes place via the consolidated database.

**Synchronizing a remote database**

When a subscription is initially set up, the two databases must be brought to a state where they both have the same set of information, ready to start replication. This process of setting up a remote database to be consistent with the consolidated database is called **synchronization**. Synchronization can be carried out manually, but the database extraction utility automates the process. You can run the extraction utility from Sybase Central or as a command-line utility.

The appropriate publication and subscription are created automatically at remote databases when you use the SQL Remote database extraction utility to create a remote database.
SQL Remote features

The following features are key to SQL Remote's design.

**Support for many subscribers** SQL Remote is designed to support replication with many subscribers to a publication.

This feature is of particular importance for mobile workforce applications, which may require replication to the laptop computers of hundreds or thousands of sales representatives from a single office database.

**Transaction log-based replication** SQL Remote replication is based on the transaction log. This enables it to replicate only changes to data, rather than all data, in each update. Also, log-base replication has performance advantages over other replication systems.

The transaction log is the repository of all changes made to a database. SQL Remote replicates changes made to databases as recorded in the transaction log. Periodically, all committed transactions in the consolidated database transaction log belonging to any publication are sent to remote databases. At remote sites, all committed transactions in the transaction log are periodically submitted to the consolidated database.

By replicating only committed transactions, SQL Remote ensures proper transaction atomicity throughout the replication setup and maintains a consistency among the databases involved in the replication, albeit with some time lag while the data is replicated.

**Central administration** SQL Remote is designed to be centrally administered, at the consolidated database. This is particularly important for mobile workforce applications, where laptop users should not have to carry out database administration tasks. It is also important in replication involving small offices that have servers but little in the way of administration resources.

Administration tasks include setting up and maintaining publications, remote users, and subscriptions, as well as correcting errors and conflicts if they occur.

**Economical resource requirements** The only software required to run SQL Remote in addition to your Adaptive Server Anywhere or Adaptive Server Enterprise DBMS is the Message Agent, and a message system. If you use the shared file link, no message system software is required as long as each remote user ID has access to the directory where the message files are stored.
Memory and disk space requirements have been kept moderate for all components of the replication system, so that you do not have to invest in extra hardware to run SQL Remote.

**Multi-platform support** The Message Agent for Adaptive Server Anywhere runs on Windows 95, Windows NT, Windows 3.x, NetWare, UNIX, and OS/2 platforms.

The Message Agent for Adaptive Server Enterprise runs on Windows NT, and UNIX.

The UNIX Message Agent supports only the shared file link.
Some sample installations

While SQL Remote can provide replication services in many different environments, its features are designed with the following characteristics in mind:

♦ SQL Remote should be a solution even when no administration load can be assigned to the remote databases, as in mobile workforce applications.

♦ Data communication among the sites may be occasional and indirect: it need not be permanent and direct.

♦ Memory and resource requirements at remote sites are assumed to be at a premium.

The following examples show some typical SQL Remote setups.

Server-to-laptop replication for mobile workforces

SQL Remote provides two-way replication between a database on an office network and personal databases on the laptop computers of sales representatives. Such a setup may use an e-mail system as a message transport.
The office server may be running a server to manage the company database. The Message Agent at the company database runs as a client application for that server.

The laptop computers may be running Windows 95 or Windows NT, and each sales representative has an Adaptive Server Anywhere personal server to manage their own data.

While away from the office, a sales representative can make a single phone call from their laptop to carry out the following functions:

- Collect new e-mail.
- Send any e-mail messages they have written.
- Collect publication updates from the office server.
- Submit any local updates, such as new orders, to the office server.

The updates may include, for example, new specials on the products the sales representative handles, or new pricing and inventory information. These are read by the Message Agent on the laptop and applied to the sales rep's database automatically, without requiring any additional action on the sales representative's part.
The new orders recorded by the sales representative are also automatically submitted to the office without any extra action on the part of the sales representative.

**Server-to-server replication among offices**

SQL Remote provides two-way replication between database servers at sales offices or outlets and a central company office, without requiring database administration experience at each sales office beyond the initial setup and that required to maintain the server.

SQL Remote is not designed for up-to-the-minute data availability at each site. Instead, it is appropriate where data can be replicated at periods of an hour or so.

Such a setup may use an e-mail system to carry the replication, if there is already a company-wide e-mail system. Alternatively, an occasional dial-up system and file transfer software can be used to implement a FILE message system.
SQL Remote is easy to configure to allow each office to receive their own set of data. Tables that are of office interest only (staff records, perhaps, if the office is a franchise) may be kept private in the same database as the replicated data.

Layers can be added to SQL Remote hierarchies: for example, each sales office server could act as a consolidated database, supporting remote subscribers who work from that office.
CHAPTER 3
Setting Up SQL Remote

About this chapter
This chapter describes how to add SQL Remote capabilities to your Adaptive Server Enterprise server.

Adaptive Server Enterprise users only
This chapter is required only for users of SQL Remote for Adaptive Server Enterprise. SQL Remote capability is automatically installed into Adaptive Server Anywhere databases.
This chapter assumes you have already installed the SQL Remote software onto your machine.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup overview</td>
<td>30</td>
</tr>
<tr>
<td>Preparing your Adaptive Server Enterprise server</td>
<td>31</td>
</tr>
<tr>
<td>Uninstalling SQL Remote</td>
<td>35</td>
</tr>
<tr>
<td>Upgrading SQL Remote for Adaptive Server Enterprise</td>
<td>36</td>
</tr>
</tbody>
</table>
Setup overview

We call the collection of databases exchanging information using SQL Remote an installation. From a physical point of view, a SQL Remote installation may consist of hundreds or even thousands of databases sharing information; but as SQL Remote keeps the information in each physical database loosely consistent at a transactional level with that in other physical databases, you can also think of the whole installation as a single dispersed database.

Deploying a large-scale SQL Remote installation can involve setting up databases on many machines. While some changes to the design and setup configuration can be made on a running installation, it is highly recommended that you deploy only when you have completed a careful analysis and test of your design.

Setup tasks

Setup of a SQL Remote installation includes the following tasks:

- **Preparing your server for SQL Remote**  You must take some steps to configure your Adaptive Server Enterprise to act as a SQL Remote site. These include installing the SQL Remote system objects and the stable queue system objects.

- **Selecting message types**  You must decide whether you want to exchange information by file sharing, e-mail, some other message type, or a combination.

- **Ensuring proper permissions are set**  Each user in the installation requires permissions on both their own database and on the consolidated database.

- **Extracting remote databases**  You must extract an initial copy of each remote database from the consolidated database.

This chapter describes each of these tasks.

All administration is at the consolidated database

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

The Sybase System Administrator should perform all SQL Remote configuration tasks. See your Adaptive Server Enterprise documentation for more information about the Adaptive Server Enterprise environment.
Preventing your Adaptive Server Enterprise server

Before you start

This section assumes the following:

♦ You have installed an Adaptive Server Enterprise server that is to contain the SQL Remote database.
♦ You have installed the SQL Remote software on your computer. To install the SQL Remote software, run the setup program from the CD-ROM.
♦ You have created a database in the Adaptive Server Enterprise server that will take part in your SQL Remote installation.
♦ You have system administrator permissions on the Adaptive Server Enterprise server, and database owner permissions in the database.

Ensuring TEMPDB is large enough

SQL Remote uses the TEMPDB database for the following purposes:

♦ The database extraction utility used to create remote databases uses TEMPDB to hold a temporary set of Adaptive Server Anywhere system tables.
♦ The Message Agent creates a temporary table called #remote when it connects to the server.

For these reasons, you should make TEMPDB larger than the 2 Mb default size. The size required depends on the number of tables and columns in your SQL Remote installation, but a size of 10 Mb is generally sufficient.

Installing SQL Remote into a database from Sybase Central

You can install SQL Remote into a database from the Sybase Central graphical administration tool. Sybase Central is available on the Windows NT and Windows 95 operating systems.

❖ To install SQL Remote into a database from Sybase Central:

1. Connect to Adaptive Server Enterprise from Sybase Central, as a user with system administrator privileges.
2. Open the Databases folder for the server.
3 Open the SQL Remote folder in the database.
4 In the right pane, double-click Setup SQL Remote, and follow the instructions in the Wizard.

Command-line installation of the SQL Remote system objects

For a database in your Adaptive Server Enterprise server to take part in a SQL Remote installation, you must install a number of SQL Remote system tables, views, and stored procedures in your database.

✦ To install the SQL Remote system objects:

1 Locate the SQL Remote initialization script ssremote.sql in your SQL Remote installation directory.
2 Make a backup copy of the ssremote.sql script file. Then add the following two lines to the beginning of ssremote.sql:

```sql
use database_name
go
```

where `database_name` is the name of the database to take part in SQL Remote replication.

These two lines set the current database to `database_name`, so that the SQL Remote tables are created in the `database_name` database. The SQL Remote tables are owned by the database owner.

3 Run the script against your Adaptive Server Enterprise server.

Change to the directory containing the script file and enter the following command line (which should be entered all on one line) to run the script:

```bash
isql -S server-name -U login_id -P password -i ssremote.sql -o logfile
```

where `server-name` is the name of the Adaptive Server Enterprise, `login_id` and `password` correspond to a user with system administrator permissions on the server who owns the database, and `logfile` is the name of a log file to hold the log information from the script.

4 Inspect the log file to confirm that the tables and procedures were created without error.

The script creates a set of SQL Remote system objects in the database.

The SQL Remote system objects

The script creates the following objects in the database:
Chapter 3  Setting Up SQL Remote

♦ SQL Remote system tables  A set of tables used to maintain SQL Remote information. These tables have names beginning with `sr_`.

♦ SQL Remote system views  A set of views that hold the SQL Remote information in a more understandable form. These views have names beginning with `sr_` and ending in `s`.

♦ SQL Remote system procedures  A set of stored procedures used to carry out SQL Remote configuration and administration tasks. These procedures have names beginning with `sp_`, indicating their system management roles.

Caution: Do not edit the SQL Remote system tables
Do not, under any circumstances, alter the SQL Remote system tables directly. Doing so may corrupt the table and make it impossible for SQL Remote to function properly. Use Sybase Central or the SQL Remote system procedures to carry out all system administration tasks.

Command-line installation of the stable queue

The stable queue is a pair of database tables that hold transactions that may still be needed by the replication system. Every Adaptive Server Enterprise database participating in a SQL Remote installation needs a stable queue.

\(~/^ For detailed information about the stable queue, see "The stable queue" on page 279.

The stable queue can exist in the same database as the database taking part in SQL Remote, or in a separate database. Keeping the stable queue in a separate database complicates the backup and recovery plan, but can improve performance by putting the stable queue workload on separate devices and/or a separate Adaptive Server Enterprise server.

❖ To install the stable queue:

1 Locate the stable queue initialization script `stableq.sql` in your SQL Remote installation directory.

2 Make a backup copy of the `stableq.sql` script file. Then add the following two lines to the beginning of `stableq.sql`:

```sql
use database_name
go
```

where `database_name` is the name of the database that will hold the stable queue.
These two lines set the current database to `database_name`, so that the stable queue is created in the `database_name` database. The stable queue tables are owned by the database owner.

3 Run the script against your Adaptive Server Enterprise server.

Change to the directory holding the stable queue script, and enter the following command line (which should be entered all on one line) to run the script:

```
isql -S server-name -U login_id -P password -i STABLEQ.SQL -o logfile
```

where `server-name` is the name of the Adaptive Server Enterprise, `login_id` and `password` correspond to a user with system administrator permissions on the server who owns the database, and `logfile` is the name of a log file to hold the log information from the script.

4 Inspect the log file to confirm that the tables and procedures were created without error.
Uninstalling SQL Remote

This section describes how to uninstall the SQL Remote objects from a database, and uninstall the stable queue from a database.

❖ To uninstall the SQL Remote objects from a database:

1. Connect to the database containing the SQL Remote objects, as a user with dbo permissions.

2. Run the `sp_drop_sql_remote` stored procedure to remove all SQL Remote objects apart from the procedure itself. The `sp_drop_sql_remote` procedure is installed along with the other SQL Remote objects.

   ```
   exec sp_drop_sql_remote
   go
   ```

3. Drop the `sp_drop_sql_remote` procedure to complete the uninstall procedure.

   ```
   drop procedure sp_drop_sql_remote
   go
   ```

❖ To uninstall the stable queue from a database:

1. Connect to the database containing the stable queue, as a user with dbo permissions.

2. Run the `sp_queue_drop` stored procedure to remove all stable queue objects apart from the procedure itself. The `sp_queue_drop` procedure is installed along with the other stable queue objects.

   ```
   exec sp_queue_drop
   go
   ```

3. Drop the `sp_queue_drop` procedure itself, to complete the uninstall procedure.

   ```
   drop procedure sp_queue_drop
   go
   ```
Upgrading SQL Remote for Adaptive Server Enterprise

This section describes the procedure for upgrading SQL Remote for Adaptive Server Enterprise.

As a SQL Remote installation may consist of a large number of databases, it is generally not practical to upgrade software on all machines at the same time. SQL Remote is designed so that upgrades can be carried out incrementally. It is not important what order SQL Remote machines are upgraded, as the message format is compatible with previous releases.

❖ To upgrade SQL Remote:

1. Back up both the consolidated database and, if it is separate, the stable queue database.
2. Install the new SQL Remote for Adaptive Server Enterprise software.
3. Run the script ssupdate.sql at the consolidated database to upgrade the SQL Remote system tables and procedures.
   The ssupdate.sql script is held in your Sybase directory.
4. Run the script squpdate.sql at the stable queue database to upgrade the SQL Remote stable queue tables and procedures.
   The squpdate.sql script is held in your Sybase directory.

The software is now upgraded.
CHAPTER 4

A Tutorial for Adaptive Server Anywhere Users

About this chapter

This chapter guides you through setting up a simple replication system.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorial overview</td>
<td>38</td>
</tr>
<tr>
<td>A tutorial using Sybase Central</td>
<td>41</td>
</tr>
<tr>
<td>Setting up a consolidated database</td>
<td>44</td>
</tr>
<tr>
<td>Set up the remote database in Sybase Central</td>
<td>49</td>
</tr>
<tr>
<td>A tutorial using Interactive SQL and DBXTRACT</td>
<td>51</td>
</tr>
<tr>
<td>Set up the consolidated database</td>
<td>53</td>
</tr>
<tr>
<td>Set up the remote database</td>
<td>57</td>
</tr>
<tr>
<td>Start replicating data</td>
<td>59</td>
</tr>
<tr>
<td>A sample publication</td>
<td>63</td>
</tr>
</tbody>
</table>
Tutorial overview

This tutorial describes how to set up a simple SQL Remote replication system using Adaptive Server Anywhere.

Tutorial goals

In the tutorial you act as the system administrator of a consolidated Adaptive Server Anywhere database, and set up a simple replication system. The replication system consists of a simple sales database, with two tables.

The consolidated database holds all of the database, while the remote database has all of one table, but only some of the rows in the other table.

The tutorial takes you through the following steps:

♦ Creating a consolidated database on your Adaptive Server Anywhere server.
♦ Creating a file-sharing replication system with a single Adaptive Server Anywhere remote database.
♦ Replicating data between the two databases.

The database

The tutorial uses a simple two-table database. One table holds information about sales representatives, and the other about customers. The tables are much simpler than you would use in a real database; this allows us to focus just on those issues important for replication.

Database schema

The database schema for the tutorial is illustrated in the figure.

<table>
<thead>
<tr>
<th>Customer</th>
<th></th>
<th>SalesRep</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_key</td>
<td>rep_key =</td>
<td>rep_key</td>
</tr>
<tr>
<td>name</td>
<td>rep_key =</td>
<td>name</td>
</tr>
<tr>
<td>rep_key</td>
<td>rep_key</td>
<td>rep_key</td>
</tr>
</tbody>
</table>

Features to note include the following:

♦ Each sales representative is represented by one row in the SalesRep table.
♦ Each customer is represented by one row in the Customer table.
Each customer is assigned to a single sales representative, and this assignment is built into the database as a foreign key from the Customer table to the SalesRep table. The relationship between the Customer table and the SalesRep table is many-to-one.

The tables in the database

The tables are described in more detail as follows:

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
</table>
| SalesRep | One row for each sales representative that works for the company. The SalesRep table has the following columns:
  ♦ rep_key An identifier for each sales representative. This is the primary key.
  ♦ name The name of each sales representative.
  The SQL statement creating this table is as follows:

  ```sql
  CREATE TABLE SalesRep ( 
    rep_key CHAR(12) NOT NULL, 
    name CHAR(40) NOT NULL, 
    PRIMARY KEY (rep_key) 
  )
  ```

| Customer | One row for each customer that does business with the company. The Customer table includes the following columns:
  ♦ cust_key An identifier for each customer. This is the primary key.
  ♦ name The name of each customer.
  ♦ rep_key An identifier for the sales representative in a sales relationship. This is a foreign key to the SalesRep table.
  The SQL statement creating this table is as follows:

  ```sql
  CREATE TABLE Customer ( 
    cust_key CHAR(12) NOT NULL, 
    name CHAR(40) NOT NULL, 
    rep_key CHAR(12) NOT NULL, 
    FOREIGN KEY (rep_key) REFERENCES SalesRep (rep_key), 
    PRIMARY KEY (cust_key) 
  )
  ```

Replication goals

The goals of the replication design are to provide each sales representative with the following information:

♦ The complete SalesRep table.
♦ Those customers assigned to them.
The tutorial describes how to meet this goal using SQL Remote.

**Sybase Central or command-line utilities**

**Use Sybase Central or the command line**

The tutorial material is presented twice. One section describes how to set up the installation using the Sybase Central management utility. The second section describes how to set up the installation using command-line utilities: this requires typing commands individually.

**Where next?**

- To work through the tutorial using Sybase Central, go to "A tutorial using Sybase Central" on page 41, next.
- To work through the tutorial entering commands explicitly, go to "A tutorial using Interactive SQL and DBXTRACT" on page 51.
A tutorial using Sybase Central

The following sections are a tutorial describing how to set up a simple SQL Remote replication system in Adaptive Server Anywhere using Sybase Central.

You do not need to enter SQL statements if you are using Sybase Central to administer SQL Remote. A tutorial for those who do not have access to Sybase Central, or who prefer to work with command-line utilities, is presented in "A tutorial using Interactive SQL and DBXTRACT" on page 51, and contains the SQL statements executed behind the scenes by Sybase Central.

In this tutorial you act as the DBA of the consolidated database, and set up a simple replication system using the file-sharing message link. The simple example is a primitive model for a sales-force automation system, with two tables. One contains a list of sales representatives, and another a list of customers. The tables are replicated in a setup with one consolidated database and one remote database. You can install this example on one computer.

This tutorial assumes that you have some familiarity with Sybase Central. If you need help with Sybase Central, use the Sybase Central online Help.

Preparing for the Sybase Central replication tutorial

This section describes the steps you need to take to prepare for the tutorial. These steps include the following:

- Create the directories and databases required for the tutorial.
- Add the tables to the consolidated database.

To prepare for the tutorial:

1. Create a directory to hold the files you make during this tutorial; for example c:\tutorial.

```
mkdir c:\tutorial
```

2. Create a subdirectory for each of the two user IDs in the replication system, to hold their messages. Create these subdirectories using the following statements at a system command line:

```
mkdir c:\tutorial\hq
mkdir c:\tutorial\field
```
The tutorial uses two databases: a consolidated database named `hq.db` and a remote database named `field.db`. At this point, you should create the `hq` database with the Create Database utility in Sybase Central:

- Start Sybase Central. You will be creating a new database so you do not have to connect to any particular existing database.
- Click Utilities in the left panel.
- Double-click Create Database in the right panel. The Create Database wizard is displayed.
- Create a database with filename `c:\tutorial\hq.db`.
- You can use the default settings for this database. Make sure you elect to maintain a transaction log. Replication cannot take place without a transaction log.

An Adaptive Server Anywhere database is simply a file, which can be copied to other locations and computers when necessary.

The next step is to add a pair of tables to the consolidated database.

**To add tables to the consolidated database:**

1. Connect to the `hq` database from Sybase Central, as user ID **DBA** using password **SQL**.
2. Click the Tables folder of the `hq` database.
3. Double-click Add Table, and use the Table Editor to create a table named **SalesRep** with the following columns:

<table>
<thead>
<tr>
<th>Key</th>
<th>Column</th>
<th>Data Type</th>
<th>Size/Prec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary key</td>
<td>rep_key</td>
<td>char</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>char</td>
<td>40</td>
</tr>
</tbody>
</table>

You do not need to use the Advanced Properties window. The columns are created by default not allowing NULL.

4. Double-click Add Table again, and use the Table Editor to create a table named **Customer** with the following columns:

<table>
<thead>
<tr>
<th>Key</th>
<th>Column</th>
<th>Data Type</th>
<th>Size/Prec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary key</td>
<td>cust_key</td>
<td>char</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>char</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>rep_key</td>
<td>char</td>
<td>5</td>
</tr>
</tbody>
</table>
Again, you do not need to use the Advanced Properties window. The columns are created by default not allowing NULL.

5 Open the Foreign Keys folder of the Customer table container, and double-click Add Foreign Key. Using the Wizard, add a foreign key to the **rep_key** column of the **SalesRep** table. You can use the default settings for this foreign key.

You are now ready for the rest of the tutorial.
Setting up a consolidated database

This section of the tutorial describes how to prepare the consolidated database of a simple replication system.

Preparing a consolidated database for replication involves the following steps:

1. Create a message type to use for replication.
2. Grant PUBLISH permissions to a user ID to identify the source of outgoing messages.
3. Grant REMOTE permissions to all user IDs that are to receive messages.
4. Create a publication describing the data to be replicated.
5. Create subscriptions describing who is to receive the publication.

You require DBA authority to carry out these tasks.

Add a SQL Remote message type

All messages sent as part of replication use a message type. A message type description has two parts:

♦ A message link supported by SQL Remote. In this tutorial, we use the FILE link.
♦ An address for this message link, to identify the source of outgoing messages.

Adaptive Server Anywhere databases already have message types created, but you need to supply an address for the message type you will use.

❖ To add an address to a message type:

1. From Sybase Central, connect to the HQ database, and open the HQ database container.
2. Click the SQL Remote folder on the left panel.
3. Double-click the Message Types folder on the right panel.
4. Double-click the FILE message type.
5. Enter a publisher address to provide a return address for remote users.

Enter the directory you have created to hold messages for the consolidated database (hq).
The address is taken relative to the SQLRemote environment variable or registry entry. As you have not set this value, the address is taken relative to the directory from which the Message Agent is run. You should run the Message Agent from your tutorial directory for the addresses to be interpreted properly.

For information about setting the SQLRemote value, see “Setting message type control parameters” on page 231.

6 Click OK to save the message type.

Add the publisher and remote user to the database

In SQL Remote's hierarchical replication system, each database may have zero or one database immediately above it (the consolidated database) and zero or more databases immediately below it (remote databases).

In this tutorial, the current database is the consolidated database of a two-level system. It has no database above it, and only one remote database below it.

The following diagram illustrates the two databases:

For any database in a SQL Remote replication setup, there are three permissions that may be granted to identify databases on the hierarchy:

- **PUBLISH permission** Identifies the current database in all outgoing messages
- **REMOTE permission** Identifies each database receiving messages from the current database that is below it on the hierarchy
- **CONSOLIDATE permission** Identifies a database receiving messages from the current database that is directly above it on the hierarchy.
Permissions can only be granted by a user with DBA authority. To carry out these examples you should connect from Sybase Central to the hq database as user ID DBA, with password SQL.

Any database, consolidated or remote, that distributes changes to other databases in the replication system is a publisher database. Each database in the replication system is identified by a single user ID. You set that ID for your database by adding a publisher to the database. This section describes setting permissions for the consolidated hq database.

First create a user ID named hq_user, who will be the publisher user ID.

相关部门

To create the publisher:

1. Click the Users & Groups folder on the left panel.
2. Double-click Add User. The New User Wizard is displayed.
3. Enter the name hq_user, with password hq_pwd, and click Next.
4. On the next page, ensure that the user is granted Remote DBA authority; this enables the user ID to run the Message Agent. Then click Next.
5. On the final page, check the box identifying this user ID as the publisher. Then click Finish to create the user.

A database can have only one publisher. You can find out who the publisher is at any time by opening the SQL Remote folder.

相关部门

Add a remote user

Each remote database is identified in the consolidated database by a user ID with REMOTE permissions. Whether the remote database is a personal database server or a network server with many users, it needs a single user ID to represent it to the consolidated database.

In a mobile workgroup setting, remote users may already be users of the consolidated database, and so no new users would need to be added; although they would need to be set as remote users.

When a remote user is added to a database, the message system they use and their address under that message system need to be stored along with their database user ID.

相关部门

To add a remote user:

1. Click the SQL Remote folder on the left panel, then click the Remote Users folder on the left panel.
3 Create a remote user with user ID field_user, password field_pwd, message type file, and address field. For the message type and address, select the FILE type and the corresponding address you are using for this user (such as field).

As with the publisher address, the address of the remote user is taken relative to the SQLRemote environment variable or registry entry. As you have not set this value, the address is taken relative to the directory from which the Message Agent is run. You should run the Message Agent from your tutorial directory for the addresses to be interpreted properly.

For information about setting the SQLRemote value, see "Setting message type control parameters" on page 231.

4 On the next page, ensure that the Send Then Close option is checked. (In many production environments you would not choose Send Then Close, but it is convenient for this tutorial.)

5 On the next page ensure that the Remote DBA authority is checked, so that the user can run the Message Agent.

6 When you have finished all the entries, click Finish to create the remote user.

You have now created the users who will use this system.

Add publications and subscriptions

This section describes how to add a publication to a database, and how to add a subscription to that publication for a user. The publication replicates all rows of the table SalesRep and some of the rows of the Customer table.

To add a publication:

1 Click the Publications folder in the SQL Remote folder.

2 Double-click Add Publication. The Publication Wizard is displayed.

3 Name the publication SalesRepData on the first page of the Wizard.

4 On the next page, click Add Table and select SalesRep from the list. Leave All Columns selected, and press OK to add the table.

Then click Add Table again, and select Customer from the list. Again, leave All Columns selected. Click the Subscribe restriction tab, and choose to Subscribe by the column rep_key. Click OK to add the table to the publication.

5 Complete the Wizard to create the publication.
Add a subscription

Each user ID that is to receive changes to a publication must have a subscription to that publication. Subscriptions can only be created for a valid remote user. You need to add a subscription to the SalesRepData publication for the remote database user field_user.

❖ To add a subscription:

1. Double-click the Publications folder, which is in the SQL Remote folder, so that the SalesRepData publication is displayed in the left panel.
2. Click the Remote Users folder so that remote users are displayed in the right panel.
3. Drag the field_user user from the right panel onto the SalesRepData publication in the left panel. The Create Subscription window is displayed. Enter a value of rep1 in the With Value box. The value rep1 is the rep_key value we will give to the user field_user in the SalesRep table.

You have now set up the consolidated database.
Set up the remote database in Sybase Central

The remote database needs to be created and configured in order to send and receive messages and participate in a SQL Remote setup.

Like the consolidated database, the remote database needs a publisher (in this case, the field_user user ID) to identify the source of outgoing messages, and it needs to have hq_user identified as a user with consolidated permissions. It needs the SalesRepData publication to be created and needs a subscription created for the hq_user user ID.

The remote database also needs to be synchronized with the consolidated database; that is, it needs to have a current copy of the data in order for the replication to start. In this case, there is no data in the publication as yet.

The database extraction utility enables you to carry out all the steps needed to create a remote database complete with subscriptions and required user IDs.

You need to extract a database from the consolidated database for remote user field_user.

To extract a database:

1. Click the Remote Users folder, which is in the SQL Remote folder.
2. Right-click the field_user remote user, and select Extract Database from the popup menu. The Extraction Wizard is displayed.
3. The first page informs you that you will extract from the running database hq.
4. On the next page, choose to Start Subscriptions Automatically, for user field_user.
5. Create the database as file c:\tutorial\field.db, using a transaction log of name field.log in the same directory.
6. Choose to extract all parts of the schema (the default setting).
7. Leave the other options at their default settings, and create the remote database.

In a proper SQL Remote setup, the remote database field would need to be loaded on to the computer using it, together with an Adaptive Server Anywhere server and any client applications required. For this tutorial, we leave the database where it is and use Interactive SQL to input and replicate data.
Set up the remote database in Sybase Central

You should connect to the field database as DBA and confirm that all the database objects are created. These include the SalesRep and Customer tables, the SalesRepData publication, and the subscription for the consolidated database.

What next? The system is now ready for replication.

For the next step, inserting and replicating data, see the section "Start replicating data" on page 59.
CHAPTER 4  A Tutorial for Adaptive Server Anywhere Users

A tutorial using Interactive SQL and DBXTRACT

The following sections are a tutorial describing how to set up a simple SQL Remote replication system for users without Windows 95 or Windows NT 3.51 or later, who cannot run Sybase Central. It may also be useful to users of Sybase Central who prefer to use command-line tools or who want to know what Sybase Central is doing behind the scenes.

This tutorial describes the SQL statements for managing SQL Remote, which can be run from Interactive SQL. It also describes how to run the dbxtract command-line utility to extract remote databases from a consolidated database.

In this tutorial you act as the DBA of the consolidated database, and set up a simple replication system using the file-sharing message link. The simple example is a primitive model for a sales-force automation system, with two tables. One contains a list of sales representatives, and another a list of customers. The tables are replicated in a setup with one consolidated database and one remote database. You can install this example on one computer.

Preparing for the replication tutorial

This section describes the steps you need to take to prepare for the tutorial. These steps include the following:

♦  Create the directories and databases required for the tutorial.
♦  Add a table to the consolidated database.

❖ To create the databases and directories for the tutorial:

1  Create a directory to hold the files you make during this tutorial; for example c:\tutorial.
   mkdir c:\tutorial

2  The tutorial uses two databases: a consolidated database named hq.db and a remote database named field.db. Change to the tutorial directory and create these databases using the following statements at a command line:

   dbinit hq.db
   dbinit field.db
The database initialization tool for Windows 3.x is \textit{dbinitw} rather than \textit{dbinit}. Under Windows 3.x, you should execute the commands from the Program Manager File►Run menu or you could make an icon for each command.

3 Create a subdirectory for each of the two user IDs in the replication system. Create these subdirectories using the following statements at a command line:

\begin{verbatim}
  mkdir c:\tutorial\hq
  mkdir c:\tutorial\field
\end{verbatim}

The next step is to add a pair of tables to the consolidated database.

\textbf{To add the tables to the consolidated database:}

1 Connect to \textit{hq.db} from Interactive SQL as user ID \textbf{DBA}, using password \textbf{SQL}.
   - Type \textbf{CONNECT} in the Interactive SQL Command window, and click \textbf{Execute}.
   - Enter the user ID \textbf{DBA} and the password \textbf{SQL}, and click \textbf{OK}.
   - If the \textit{hq} database is already running, type \textit{hq} in the Database Name box. If it is not running, enter the Database File \texttt{c:\tutorial\hq.db}. Then click \textbf{OK} to connect.

2 Enter the following CREATE TABLE statement to create the \textbf{SalesRep} table:

\begin{verbatim}
CREATE TABLE SalesRep {
  rep_key CHAR(12) NOT NULL,
  name CHAR(40) NOT NULL,
  PRIMARY KEY ( rep_key )
};
\end{verbatim}

3 Enter the following CREATE TABLE statement to create the \textbf{Customer} table:

\begin{verbatim}
CREATE TABLE Customer {
  cust_key CHAR(12) NOT NULL,
  name CHAR(40) NOT NULL,
  rep_key CHAR(12) NOT NULL,
  FOREIGN KEY REFERENCES SalesRep,
  PRIMARY KEY ( cust_key )
};
\end{verbatim}

You are now ready for the rest of the tutorial.
Set up the consolidated database

This section of the tutorial describes how to set up the consolidated database of a simple replication system.

You require DBA authority to carry out this task.

Create a SQL Remote message type

All messages sent as part of replication use a message type. A message type description has two parts:

♦ A message link supported by SQL Remote. In this tutorial, we use the FILE link.
♦ An address for this message link, to identify the source of outgoing messages.

❖ To create the message type:

♦ Create the file message type using the following statement:

```
CREATE REMOTE MESSAGE
TYPE file
ADDRESS 'hq'
```

The address (hq) for a file link is a directory in which files containing the message are placed. It is taken relative to the SQLRemote environment variable or registry entry. As you have not set this value, the address is taken relative to the directory from which the Message Agent is run. You should run the Message Agent from your tutorial directory for the addresses to be interpreted properly.

❖ For information about setting the SQLRemote value, see "Setting message type control parameters" on page 231.

Grant PUBLISH and REMOTE at the consolidated database

In the hierarchical replication system supported by SQL Remote, each database may have one consolidated database immediately above it in the hierarchy and many databases immediately below it on the hierarchy (remote databases).

PUBLISH permission identifies the current database for outgoing messages, and the REMOTE permission identifies each database receiving messages from the current database.
Permissions can only be granted by a user with DBA authority. To carry out these examples you should connect using the Interactive SQL utility to hq as user ID DBA, with password SQL.

Each database that distributes its changes to other databases in the replication system is a publisher database. Each database in the replication system that publishes changes to a database is identified by a single user ID. You set that ID for your database using the GRANT PUBLISH statement. This section describes setting permissions for the consolidated database (hq.db).

❖ To create a publisher for the database:

❖ Connect to the database using Interactive SQL, and type the following statement:

```
GRANT CONNECT
TO hq_user
IDENTIFIED BY hq_pwd ;
```

```
GRANT PUBLISH TO hq_user ;
```

You can check the publishing user ID of a database at any time using the CURRENT PUBLISHER special constant:

```
SELECT CURRENT PUBLISHER
```

Each remote database is identified using the GRANT REMOTE statement. Whether the remote database is a personal server or a network server with many users, it needs a single user ID to represent it to the consolidated database.

In a mobile workgroup setting, remote users may already be users of the consolidated database, and so this would require no extra action on the part of the DBA.

The GRANT REMOTE statement identifies the message system to be used when sending messages to the recipient, as well as the address.

❖ To add a remote user:

❖ Connect to the database using Interactive SQL, and enter the following statements:

```
GRANT CONNECT TO field_user
IDENTIFIED BY field_pwd ;
```

```
GRANT REMOTE TO field_user
TYPE file ADDRESS 'field' ;
```
The address string is the directory used to hold messages for field_user, enclosed in single quotes. It is taken relative to the SQLRemote environment variable or registry entry. As you have not set this value, the address is taken relative to the directory from which the Message Agent is run. You should run the Message Agent from your tutorial directory for the addresses to be interpreted properly.

For information about setting the SQLRemote value, see "Setting message type control parameters" on page 231.

**Create publications and subscriptions**

A publication is created using a CREATE PUBLICATION statement. This is a data definition language statement, and requires DBA authority. For the tutorial, you should connect to the hq database as user ID DBA, password SQL, to create a publication.

Create a publication named SalesRepData, which replicates all rows of the table SalesRep, and some of the rows of the table Customer.

- **To create the publication:**
  - Connect to the database from Interactive SQL, and enter the following statement:
    ```sql
    CREATE PUBLICATION SalesRepData {
      TABLE SalesRep,
      TABLE Customer SUBSCRIBE BY rep_key
    }
    ```

- **To create the subscription:**
  - Connect to the database from Interactive SQL, and enter the following statement:
    ```sql
    CREATE SUBSCRIPTION
    TO SalesRepData ('rep1')
    FOR field_user ;
    ```
    The value rep1 is the rep_key value we will give to the user field_user in the SalesRep table.
Set up the consolidated database

The full CREATE SUBSCRIPTION statement allows control over the data in subscriptions; allowing users to receive only some of the rows in the publication. For more information, see "CREATE SUBSCRIPTION statement" on page 360.

The CREATE SUBSCRIPTION statement identifies the subscriber and defines what they receive. However, it does not synchronize data, or start the sending of messages.
Set up the remote database

The remote database needs to be configured in order to send and receive messages and participate in a SQL Remote setup. Like the consolidated database, the remote database needs a CURRENT PUBLISHER to identify the source of outgoing messages, and it needs to have the consolidated database identified as a subscriber. The remote database also needs the publication to be created and needs a subscription created for the consolidated database. The remote database also needs to be synchronized with the consolidated database; that is, it needs to have a current copy of the data in order for the replication to start.

The dbxtract utility enables you to carry out all the steps needed to create a remote database complete with subscriptions and required user IDs.

Extract the remote database information

Leave the hq database running, and change to the tutorial directory.

Type the following command at the system command line (all on one line) to extract a database for the user field_user from the consolidated database:

```
  dbxtract -v -c "dbname=hq;uid=dba;pwd=sql" c:\tutorial field_user
```

In Windows 3.x, you should run the following command either from File►Run in Program Manager or from an icon:

```
  dbxtractw -v -c "dbname=hq;uid=dba;pwd=sql" c:\tutorial field_user
```

The -v command-line switch produces more verbose output. This is useful during development.

This command assumes the hq database is currently running on the default server. If the database is not running, you should enter a database file parameter in the connection string:

```
  dbf=hq.db
```

instead of the dbname database name parameter.

For details of the dbxtract utility and its options, see "The extraction command-line utility" on page 315.

The dbxtract command creates a SQL command file named reload.sql in the current directory and a data file in the c:\tutorial directory. It also starts the subscriptions to the remote user.

The next step is to load these files into the remote database.
Load the remote database information

To load the database information:

1. From the tutorial directory, connect to the remote database field.db from Interactive SQL as user ID DBA and password SQL.
   
   You can do this by typing CONNECT in the Interactive SQL command window, and then entering DBA, SQL, and the database file c:\tutorial\field.db in the connection dialog box.

2. Once connected, run the reload.sql command file:

   ```sql
   READ C:\tutorial\reload.sql
   ```

   The reload.sql command file carries out the following tasks:
   
   ♦ Creates a message type at the remote database.
   ♦ Grants PUBLISH and REMOTE permissions to the remote and consolidated database, respectively.
   ♦ Creates the table in the database. If the table had contained any data before extraction, the command file would fill the replicated table with a copy of the data.
   ♦ Creates a publication to identify the data being replicated.
   ♦ Creates the subscription for the consolidated database, and starts the subscription.

   While connected to the field database as DBA, confirm that the tables are created by typing

   ```sql
   SELECT * FROM SalesRep ;
   ```

   ```sql
   SELECT * FROM Customer ;
   ```

What next?

The system is now ready for replication.

☞ For the next step, inserting and replicating data, see the section "Start replicating data" on page 59.
Start replicating data

You now have a replication system in place. In this section, data is replicated from the consolidated database to the remote database, and from the remote to the consolidated database.

Enter data at the consolidated database

First, enter some data into the consolidated database.

❖ To enter data at the consolidated database:

1. Connect to the consolidated database hq from the Interactive SQL utility as user ID DBA, with password SQL.

2. Enter and commit two rows into the SalesRep table:

   ```sql
   INSERT INTO SalesRep (rep_key, name) 
   VALUES ('rep1', 'Field User'); 
   INSERT INTO SalesRep (rep_key, name) 
   VALUES ('rep2', 'Another User'); 
   COMMIT;
   ```

3. Enter and commit two rows into the Customer table:

   ```sql
   INSERT INTO Customer (cust_key, name, rep_key) 
   VALUES ('cust1', 'Ocean Sports', 'rep1'); 
   INSERT INTO Customer (cust_key, name, rep_key) 
   VALUES ('cust2', 'Sports Plus', 'rep2'); 
   COMMIT;
   ```

4. Confirm that the data is entered by viewing the data in the tables:

   ```sql
   SELECT * FROM SalesRep;
   SELECT * FROM Customer;
   ```

The next step is to send the relevant rows to the remote database.

Send data from the consolidated database

To send the rows to the remote database, you must run the Message Agent at the consolidated database. The dbremote program is the Message Agent for Adaptive Server Anywhere.
To send the data to the remote database:

1. From a command prompt, change to your tutorial directory. For example,
   
   ```
   > c:
   > cd c:\tutorial
   ```

2. Enter the following statement at the command line to run the Message Agent against the consolidated database:
   ```
   dbremote -c "dbname=hlq;uid=dba;p=sql"
   ```

   In Windows 3.x, you should run the following command either from File ➤ Run in Program Manager or from an icon:
   ```
   dbremote w -c "dbname=hlq;uid=dba;p=sql"
   ```

   with the tutorial directory as the working directory. These command lines assume that the hlq database is currently running on the default server.

   For more information on `dbremote` command line switches, see "The Message Agent" on page 306.

3. Click Shutdown on the Message Agent window to stop the Message Agent when the messages have been sent. The Message Agent window displays the message Execution completed when all processing is complete.

Receive data at the remote database

To receive the insert statement at the remote database, you must run the Message Agent, `dbremote`, at the remote database.

To receive data at the remote database:

1. From a command prompt, change to your tutorial directory. For example,
   
   ```
   > c:
   > cd c:\tutorial
   ```

2. Enter the following statement at the command line to run the Message Agent against the field database:
   ```
   dbremote -c "dbname=field;uid=dba;p=sql"
   ```

   In Windows 3.x, you should run the following command either from File ➤ Run in Program Manager or from an icon:
   ```
   dbremotw -c "dbname=field;uid=dba;p=sql"
   ```
These command lines assume that the field database is currently running on the default server.

For more information on dbremote command line switches, see "The Message Agent" on page 306.

3 Click Shutdown on the Message Agent window to stop the Message Agent when the messages have been processed. The Message Agent window displays the message Execution completed when all processing is complete.

The Message Agent window displays status information while running. This information can be output to a log file for record keeping in a real setup. You will see that the Message Agent first receives a message from HQ, and then sends a message. This return message contains confirmation of successful receipt of the replication update; such confirmations are part of the SQL Remote message tracking system that ensures message delivery even in the event of message system errors.

To verify that the data has arrived:

1 Connect to the field database using Interactive SQL.

2 Inspect the SalesRep table by typing the following statement:

   SELECT * FROM SalesRep

   You will see that the SalesRep table contains both rows entered at the consolidated database. This is because the SalesRepData publication included all the data from the SalesRep table.

3 Inspect the SalesRep table by typing the following statement:

   SELECT * FROM Customer

   You will also see that the Customer table contains only row (Ocean Sports) entered at the consolidated database. This is because the SalesRepData publication included only those customers assigned to the subscribed Sales Rep.

Replicate from the remote database to the consolidated database

You should now try entering data at the remote field database and sending it to the consolidated database. Only the outlines are presented here.
To replicate data from the remote database to the consolidated database:

1. Connect to the **field** database from Interactive SQL.
2. INSERT a row at the remote database. For example
   
   ```
   INSERT INTO Customer (cust_key, name, rep_key) VALUES ('cust3', 'North Land Trading', 'rep1')
   ```

3. COMMIT the row:
   
   ```
   COMMIT;
   ```

4. With the **field.db** database running, run `dbremote` to send the message to the consolidated database.
   
   ```
   dbremote -c "dbn=field;uid=dba;pwd=sql"
   ```
   (For Windows 3.x, run the `dbremotw` equivalent.)

5. With the **hq.db** database running, run `dbremote` to receive the message at the consolidated database:
   
   ```
   dbremote -c "dbn=hq;uid=dba;pwy=sql"
   ```

6. Connect to the consolidated database and display the **Customer** table:
   
   ```
   SELECT *
   FROM Customer
   ```

<table>
<thead>
<tr>
<th>cust_key</th>
<th>name</th>
<th>rep_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust1</td>
<td>Ocean Sports</td>
<td>rep1</td>
</tr>
<tr>
<td>cust2</td>
<td>Sports Plus</td>
<td>rep2</td>
</tr>
<tr>
<td>cust3</td>
<td>North Land Trading</td>
<td>rep1</td>
</tr>
</tbody>
</table>

In this simple example, there is no protection against duplicate entries of primary key values. SQL Remote does provide for such protection. For information, see the chapters on SQL Remote Design.
A sample publication

The command file `salespub.sql` contains a set of statements that creates a publication on the sample database. This publication illustrates several of the points of the tutorial, in more detail.

- To add the publication to the sample database:
  1. Connect to the sample database from Interactive SQL.
  2. Type `READ path\scripts\salespub.SQL`, where `path` is your Adaptive Server Anywhere installation directory.

The `salespub.sql` publication adds columns to some of the tables in the sample database, creates a publication and subscriptions, and also adds triggers to resolve update conflicts that may occur.
A sample publication
CHAPTER 5

A Tutorial for Adaptive Server Enterprise Users

About this chapter

This chapter presents a tutorial in which you set up a simple SQL Remote replication system between an Adaptive Server Enterprise database and an Adaptive Server Anywhere database, from scratch.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>66</td>
</tr>
<tr>
<td>Setting up SQL Remote using Sybase Central</td>
<td>69</td>
</tr>
<tr>
<td>Setting up the consolidated database</td>
<td>72</td>
</tr>
<tr>
<td>Set up the remote database in Sybase Central</td>
<td>77</td>
</tr>
<tr>
<td>A tutorial using isql and ssxtract</td>
<td>79</td>
</tr>
<tr>
<td>Setting up the consolidated database</td>
<td>82</td>
</tr>
<tr>
<td>Start replicating data</td>
<td>88</td>
</tr>
</tbody>
</table>
Introduction

This chapter presents a tutorial to lead you through setting up a SQL Remote installation. The installation replicates data between an Adaptive Server Enterprise database (the consolidated database) and an Adaptive Server Anywhere database (the remote database).

Tutorial goals

In the tutorial you act as the system administrator of a consolidated Adaptive Server Enterprise database, and set up a simple replication system. The replication system consists of a simple sales database, with two tables.

The consolidated database holds all of the database, while the remote database has all of one table, but only some of the rows in the other table.

The tutorial takes you through the following steps:

♦ Creating a consolidated database on your Adaptive Server Enterprise server.

♦ Creating a file-sharing replication system with a single Adaptive Server Anywhere remote database.

♦ Replicating data between the two databases.

The database

The tutorial uses a simple two-table database. One table holds information about sales representatives, and the other about customers. The tables are much simpler than you would use in a real database; this allows us to focus just on those issues important for replication.

Database schema

The database schema for the tutorial is illustrated in the figure.

Features to note include the following:

♦ Each sales representative is represented by one row in the SalesRep table.

♦ Each customer is represented by one row in the customer table.
Each customer is assigned to a single Sales representative, and this assignment is built into the database as a foreign key from the Customer table to the SalesRep table. The relationship between the Customer table and the SalesRep table is many-to-one.

The tables are described in more detail as follows:

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SalesRep</td>
<td>One row for each sales representative that works for the company. The SalesRep table has the following columns:</td>
</tr>
<tr>
<td></td>
<td>♦ <strong>rep_key</strong> An identifier for each sales representative. This is the primary key.</td>
</tr>
<tr>
<td></td>
<td>♦ <strong>name</strong> The name of each sales representative.</td>
</tr>
<tr>
<td></td>
<td>The SQL statement creating this table is as follows:</td>
</tr>
<tr>
<td></td>
<td>CREATE TABLE SalesRep (</td>
</tr>
<tr>
<td></td>
<td>rep_key CHAR(12) NOT NULL,</td>
</tr>
<tr>
<td></td>
<td>name CHAR(40) NOT NULL,</td>
</tr>
<tr>
<td></td>
<td>PRIMARY KEY (rep_key)</td>
</tr>
<tr>
<td>Customer</td>
<td>One row for each customer that does business with the company. The Customer table includes the following columns:</td>
</tr>
<tr>
<td></td>
<td>♦ <strong>cust_key</strong> An identifier for each customer. This is the primary key.</td>
</tr>
<tr>
<td></td>
<td>♦ <strong>name</strong> The name of each customer.</td>
</tr>
<tr>
<td></td>
<td>♦ <strong>rep_key</strong> An identifier for the sales representative in a sales</td>
</tr>
<tr>
<td></td>
<td>relationship. This is a foreign key to the SalesRep table.</td>
</tr>
<tr>
<td></td>
<td>The SQL statement creating this table is as follows:</td>
</tr>
<tr>
<td></td>
<td>CREATE TABLE Customer (</td>
</tr>
<tr>
<td></td>
<td>cust_key CHAR(12) NOT NULL,</td>
</tr>
<tr>
<td></td>
<td>name CHAR(40) NOT NULL,</td>
</tr>
<tr>
<td></td>
<td>rep_key CHAR(12) NOT NULL,</td>
</tr>
<tr>
<td></td>
<td>FOREIGN KEY (rep_key) REFERENCES SalesRep (rep_key),</td>
</tr>
<tr>
<td></td>
<td>PRIMARY KEY (cust_key)</td>
</tr>
</tbody>
</table>

Replication goals

The goals of the replication design are to provide each sales representative with the following information:

♦ The complete SalesRep table.
♦ Those customers assigned to them.
Introduction

The tutorial describes how to meet this goal using SQL Remote.

**Use Sybase Central or the command line**

The tutorial material is presented twice. One section describes how to set up the installation using the Sybase Central management utility. The second section describes how to set up the installation using the Adaptive Server Enterprise and Adaptive Server Anywhere interactive SQL utilities: this requires typing commands individually.

**Where next?**

- To work through the tutorial using Sybase Central, go to "Setting up SQL Remote using Sybase Central" on page 69.
- To work through the tutorial entering commands explicitly, go to "A tutorial using isql and ssxtract" on page 79.
Setting up SQL Remote using Sybase Central

The following sections are a tutorial describing how to set up a simple SQL Remote replication system using Sybase Central.

You do not need to enter SQL statements if you are using Sybase Central to administer SQL Remote. A tutorial for those who do not have access to Sybase Central, who prefer to use command-line utilities, or who wish to understand the individual commands executed, is presented in "A tutorial using isql and ssxtract" on page 79. The commands described in that section are executed behind the scenes by Sybase Central.

First steps

Create a login name and password
To work through the tutorial, you must have system administrator privileges on an Adaptive Server Enterprise server. The tutorial assumes that your login name is the two-letter word sa and that this login name has a password of sysadmin.

Create a database
Create a database named hq on your Adaptive Server Enterprise with sufficient space to hold the tables and data required by the tutorial database. A space of 10 Mb is sufficient.

❖ To create a database from Sybase Central:

1. Connect to the Adaptive Server Enterprise from Sybase Central.
2. Open the Databases folder, and double-click Add database in the right pane.
3. Enter the name hq on the first page of the Wizard.
4. Follow the instructions in the Wizard.

☞ For information on how to create databases and assign space to them, see your Adaptive Server Enterprise documentation.

Install SQL Remote
You need to install SQL Remote into the hq database.

❖ To install SQL Remote into the HQ database:

1. Open the hq database container, on the left pane of Sybase Central.
2. Open the SQL Remote folder, double-click Setup SQL Remote, and follow the instructions. For this tutorial, you should install the stable queue in the hq database.
If your TEMPDB database is too small, you may have to add space to it for this to work.

For a full description of how to install SQL Remote, see "Setting Up SQL Remote" on page 29.

Create directories for messages

Make a directory for the files created in this tutorial. For example:

```
mkdir c:\tutorial
```

You should create a directory for each of the two users of the replication system under your tutorial directory:

```
mkdir c:\tutorial\hq
mkdir c:\tutorial\field
```

The next step is to add a pair of tables to the consolidated database.

❖ To add tables to the consolidated database:

1. Connect to the **hq** database from Sybase Central, as a system administrator.

2. Click the User Tables folder of the **hq** database.

3. Double-click Add Table, and use the Table Editor to create a table named **SalesRep** with the following columns:

<table>
<thead>
<tr>
<th>Key</th>
<th>Column</th>
<th>Data Type</th>
<th>Size/Prec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary key</td>
<td>rep_key</td>
<td>char</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>char</td>
<td>40</td>
</tr>
</tbody>
</table>

You do not need to use the Advanced Properties window. The columns are created by default not allowing NULL.

4. Double-click Add Table again, and use the Table Editor to create a table named **Customer** with the following columns:

<table>
<thead>
<tr>
<th>Key</th>
<th>Column</th>
<th>Data Type</th>
<th>Size/Prec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary key</td>
<td>cust_key</td>
<td>char</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>char</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>rep_key</td>
<td>char</td>
<td>5</td>
</tr>
</tbody>
</table>

Again, you do not need to use the Advanced Properties window. The columns are created by default not allowing NULL.
5  Open the Foreign Keys folder of the Customer table container, and 
double-click Add Foreign Key. Using the Wizard, add a foreign key to 
the rep_key column of the SalesRep table. You can use the default 
settings for this foreign key.

You are now ready for the rest of the tutorial.
Setting up the consolidated database

This section of the tutorial describes how to prepare the consolidated database of a simple replication system.

Preparing a consolidated database for replication involves the following steps:

1. Create a message type to use for replication.
2. Grant PUBLISH permissions to a user ID to identify the source of outgoing messages.
3. Grant REMOTE permissions to all user IDs that are to receive messages.
4. Create a publication describing the data to be replicated.
5. Create subscriptions describing who is to receive the publication.

You should have system administrator authority to carry out these tasks.

Add a SQL Remote message type

All messages sent as part of replication use a message type. A message type description has two parts:

- A message link supported by SQL Remote. In this tutorial, we use the FILE link.
- An address for this message link, to identify the source of outgoing messages.

Adaptive Server Anywhere databases already have message types created, but you need to supply an address for the message type you will use.

❖ To add an address to a message type:

1. From Sybase Central, open the hq database container.
2. Click the SQL Remote folder on the left panel.
3. Double-click the Message Types folder on the right panel.
4. Double-click the file message type.
5. Enter a publisher address to provide a return address for remote users. Enter hq: the directory you have created to hold messages for the consolidated database.
The address (hq) for a file link is a directory in which files containing the message are placed. It is taken relative to the SQLRemote environment variable or registry entry. As you have not set this value, the address is taken relative to the directory from which the Message Agent is run. You should run the Message Agent from your tutorial directory for the addresses to be interpreted properly.

For information about setting the SQLRemote value, see "Setting message type control parameters" on page 231.

6 Click OK to save the message type.

Create the necessary users and permissions

A set of users and permissions are required for SQL Remote installations. In this tutorial, the following are required:

♦ A publisher user name.
♦ A remote user or subscriber.

This section describes the steps you need to take to create each user and assign them the necessary permissions.

To create the publisher:

1 From Sybase Central, open the container for the hq database.
2 Open the Users folder, and double-click Add User.
3 Create a user named hq_user, mapping the login name to an available login name on your server. In this tutorial, a password of hq_pwd is assumed.
4 Make this user the publisher of the HQ database. Open the SQL Remote folder, and double-click Set Publisher. Select hq_user from the list of users to set it as the publisher.

A database can have only one publisher. You can find out who the publisher is at any time by opening the SQL Remote folder.

Add a remote user

Each remote database is identified in the consolidated database by a user ID with REMOTE permissions. Whether the remote database is a personal server or a network server with many users, it needs a single user ID to represent it to the consolidated database.

In a mobile workgroup setting, remote users may already be users of the consolidated database, and so no new users would need to be added; although they would need to be set as remote users.
Setting up the consolidated database

When a remote user is added to a database, the message system they use and their address under that message system need to be stored along with their database user ID.

❖ **To add a remote user:**

1. Double click the SQL Remote folder on the left panel, then click the Remote Users folder on the left panel.

2. If you do not have a login name that you can use for the remote user, open the logins folder directly under the server container, and add a login. The name is unimportant.

   Double-click Add Remote User on the right panel. The New Remote User wizard is displayed.

3. Create a remote user with name **field_user**. For the message type and address, select the FILE type and the corresponding address you are using for this user (such as *field*).

   As with the publisher address, the address of the remote user (*field*) is a directory relative to the SQLRemote environment variable or registry entry. As you have not set this value, the address is taken relative to the directory from which the Message Agent is run. You should run the Message Agent from your tutorial directory for the addresses to be interpreted properly.

   Note: For information about setting the SQLRemote value, see "Setting message type control parameters" on page 231.

4. On the next page, ensure that the Send Then Close option is checked. (In many production environments you would not choose Send Then Close, but it is convenient for this tutorial.)

5. When you have finished all the entries, click Finish to create the remote user.

You have now created the users who will use this system.

Create the publication and subscription

This section describes how to add a publication to a database, and how to add a subscription to that publication for a user. The publication replicates all rows of the table **SalesRep** and some of the rows of the **Customer** table.

The first step is to mark the **SalesRep** and **Customer** tables for SQL Remote replication. Marking a table for SQL Remote replication enables it to be included in publications.
To mark the tables for SQL Remote replication:
1. Open the SQL Remote folder in the *hq* database.
2. Open the Remote Tables folder, and double-click Add Remote Table.
3. Select SalesRep from the list of tables. You can leave the Conflict Resolution fields as they are. Click Apply to mark the table for SQL Remote replication.
4. Select Customer from the list of tables. Again, you can leave the Conflict Resolution fields as they are. Click OK to mark the table for SQL Remote replication.

To add a publication:
1. Click the Publications folder in the SQL Remote folder.
2. Double-click Add Publication. The Publication Wizard is displayed.
3. Name the publication SalesRepData on the first page of the Wizard.
4. On the next page, click Add Table and select SalesRep from the list. Leave All Columns selected, and press OK to add the table. Then click Add Table again, and select Customer from the list. Again, leave All Columns selected. Click the Subscribe Restriction tab, and choose to Subscribe by the column rep_key. Click OK to add the table to the publication.
5. Complete the Wizard to create the publication.

Add a subscription: Each user ID that is to receive changes to a publication must have a subscription to that publication. Subscriptions can only be created for a valid remote user. You need to add a subscription to the SalesRepData publication for the remote database user field_user.

To add a subscription:
1. Double-click the Publications folder, which is in the SQL Remote folder, so that the SalesRepData publication is displayed in the left panel.
2. Click the Remote Users folder so that remote users are displayed in the right panel.
3. Drag the field_user user from the right panel onto the SalesRepData publication in the left panel. The Create Subscription window is displayed. Enter a value of rep1 in the With Value box. The value rep1 is the rep_key value we will give to the user field_user in the SalesRep table.
Setting up the consolidated database

At this stage, the subscription is not **started**—that is, no data will be exchanged. The subscription is started by the database extraction utility.

You have now set up the consolidated database.
Set up the remote database in Sybase Central

The remote database needs to be created and configured in order to send and receive messages and participate in a SQL Remote setup.

Like the consolidated database, the remote database needs a publisher (in this case, the field_user user ID) to identify the source of outgoing messages, and it needs to have hq_user identified as a user with consolidated permissions. It needs the SalesRepData publication to be created and needs a subscription created for the hq_user user ID.

The remote database also needs to be synchronized with the consolidated database; that is, it needs to have a current copy of the data in order for the replication to start. In this case, there is no data in the publication as yet.

The database extraction utility enables you to carry out all the steps needed to create a remote database complete with subscriptions and required user IDs.

You need to extract a database from the consolidated database for remote user field_user.

❖ **To extract a database:**

1. Click the Remote Users folder, which is in the SQL Remote folder.
2. Right-click the field_user remote user, and select Extract Database from the popup menu. The Extraction Wizard is displayed.
3. Use the user ID and password that you have used to create the tables and users in the database.
4. On the next page, check Start Subscriptions Automatically, for user field_user. Also, check Create New Remote Database. Adaptive Server Anywhere must be installed for Create New Remote Database to be available.
5. Create the database as file c:\tutorial\field.db, using a transaction log in the same directory.
6. Choose to extract all parts of the schema (the default setting). Leave the other options at their default settings, and create the remote database.

You should connect to the field database as DBA and confirm that all the database objects are created.

❖ **To connect to and inspect the remote database:**

1. From Sybase Central, choose Tools→Connect→Sybase Adaptive Server Anywhere.
2 Provide the user ID **DBA** and the password **SQL**. These must be typed in upper case, as the database was created as case sensitive. Select the field.db file, and connect.

3 Open the database container, and confirm that the tables and user names are present.

In a proper SQL Remote setup, the remote database field would need to be loaded on to the computer using it, together with an Adaptive Server Anywhere server and any client applications required. For this tutorial, we leave the database where it is and use isql to input and replicate data.

**What next?**

The system is now ready for replication.

☞ For the next step, inserting and replicating data, see the section "Start replicating data" on page 88.
A tutorial using isql and ssxtract

The following sections are a tutorial describing how to set up a simple SQL Remote replication system. This version of the tutorial does not use Sybase Central.

This tutorial describes the stored procedures used to configure and manage SQL Remote. It also describes how to run the ssxtract command-line utility to extract remote databases from a consolidated database and the Message Agents to send information between the databases in the replication system.

In this tutorial you act as the administrator of the consolidated database, and set up a simple replication system using the file-sharing message link. The simple example is a primitive model for a sales-force automation system, with two tables. One contains a list of sales representatives, and another a list of customers. The tables are replicated in a setup with one consolidated database and one remote database. You can install this example on one computer.

First steps

Create a login name and password

To work through the tutorial, you must have system administrator privileges on an Adaptive Server Enterprise server. The tutorial assumes that your login name is the two-letter word sa and that this login name has a password of sysadmin.

The tutorial uses the Adaptive Server Enterprise isql utility. With the login name and password as given above, you can connect to your Adaptive Server Enterprise server using the following command line:

```
isql -S server-name -U sa -P sysadmin
```

where server-name is the name of the Adaptive Server Enterprise to which you connect.

Ensure that you have an appropriate login ID and can connect to your server before starting this tutorial.

Create a database

Create a database named hq on your Adaptive Server Enterprise server with sufficient space to hold the tables and data required by the tutorial database. A space of 4 Mb is sufficient.

- **To create a database:**

  1. Using isql, connect to the server as a user with system administrator privileges:

  ```
isql -S server-name -U sa -P sysadmin
  ```
2 Use the master database:
   
   ```
   use master
   go
   ```

3 Create a database named `hq`. In this example, we use a 5 Mb database with a 5 Mb log, on two different devices:

   ```
   create database hq
   on database_device = 5
   log on log_device = 5
   go
   ```

For more information on how to create databases and assign space to them, see your Adaptive Server Enterprise documentation.

You need to install SQL Remote into the `hq` database.

To install SQL Remote into the `hq` database:

1 If the system administrator login name you are using does not have the `hq` database as the default database, make a backup copy of the `ssremote.sql` script from your installation directory, and add the following two lines to the beginning of the script:

   ```
   use hq
   go
   ```

2 Change to the tutorial directory. Then, using `isql`, connect to the server using the `hq` database, and run the `ssremote.sql` script from your SQL Remote installation directory. The following command should be entered all on one line:

   ```
   isql -S server-name -U sa -P sysadmin -i ssremote.sql
   ```

3 If the system administrator login name you are using does not have the `hq` database as the default database, make a backup copy of the `stableq.sql` script from your installation directory, and add the following two lines to the beginning of the script:

   ```
   use hq
   go
   ```

4 Using `isql`, connect to the server using the `hq` database, and run the `stableq.sql` script from your SQL Remote installation directory. The following command should be entered all on one line:

   ```
   isql -S server-name -U sa -P sysadmin -i stableq.sql
   ```

Create directories for messages: Create a directory to hold the files from this tutorial. For example:

   ```
   mkdir c:\tutorial
   ```
You should create a directory for each of the two users of the replication system under your parent directory for this tutorial:

```
mkdir c:\tutorial\hq
mkdir c:\tutorial\field
```

The next step is to add a pair of tables to the consolidated database.

✔ **To add tables to the consolidated database:**

1  Connect to the `hq` database from `isql`, as a system administrator.
2  Use the `hq` database:
   ```
   use hq
   go
   ```
3  Create the `SalesRep` table with the following statement:
   ```
   create table SalesRep (
      rep_key char(12) not null,
      name char(40) not null,
      primary key (rep_key) 
   )
   go
   ```
4  Create the `Customer` table with the following statement:
   ```
   create table Customer (
      cust_key char(12) not null,
      name char(40) not null,
      rep_key char(12) not null,
      primary key (cust_key) 
   )
   go
   ```
5  Alter the `Customer` table to add a foreign key to the `SalesRep` table:
   ```
   alter table Customer
   add foreign key
   ( rep_key ) references SalesRep
   go
   ```

You are now ready for the rest of the tutorial.
Setting up the consolidated database

This section of the tutorial describes how to prepare the consolidated database of a simple replication system.

Preparing a consolidated database for replication involves the following steps:

1. Create a message type to use for replication.
2. Grant PUBLISH permissions to a user ID to identify the source of outgoing messages.
3. Grant REMOTE permissions to all user IDs that are to receive messages.
4. Create a publication describing the data to be replicated.
5. Create subscriptions describing who is to receive the publication.

You should have system administrator authority to carry out these tasks.

Create the message links and addresses

In this tutorial, messages are exchanged using the shared file link. You must create a FILE message type supplying the address of the consolidated database publisher.

❖ To create the message type:

❖ Execute the **sp_remote_type** stored procedure, using HQ as the address of the consolidated database publisher:

```
sp_remote_type file, hq
go
```

The address (hq) for a file link is a directory in which files containing the message are placed. It is taken relative to the SQLRemote environment variable or registry entry. As you have not set this value, the address is taken relative to the directory from which the Message Agent is run. You should run the Message Agent from your tutorial directory for the addresses to be interpreted properly.

❖ For information about setting the SQLRemote value, see "Setting message type control parameters" on page 231.

With the message type defined, you can now make the necessary users.
Create the necessary users and permissions

A set of users and permissions are required for SQL Remote installations. In this tutorial, the following are required:

♦ A remote user or subscriber, with name **field_user**.
♦ A publisher user name, called **hq_user**.

This section describes the steps you need to take to create each user and assign them the necessary permissions.

vê To create the publisher:

1 Add a login called **hq_user**, with **hq** as the default database and with system administrator access:

   ```
   exec sp_addlogin hq_user, hq_pwd, hq
   go
   exec sp_role 'grant', sa_role, hq_user
   go
   ```

2 Add the login name as a user to the HQ database:

   ```
   use hq
   go
   exec sp_adduser hq_user
   go
   ```

3 Make this user the publisher of the HQ database:

   ```
   exec sp_publisher hq_user
   go
   ```

Add a remote user

Each remote database is identified in the consolidated database by a user ID with REMOTE permissions. Whether the remote database is a single-user server or a database server with many users, it needs a single user ID to represent it to the consolidated database.

In a mobile workgroup setting, remote users may already be users of the consolidated database, and so no new users would need to be added; although they would need to be set as remote users.

When a remote user is added to a database, the message system they use and their address under that message system need to be stored along with their database user ID.

vê To create the subscriber:

1 If you do not have a login name that you can use for the remote user, add a login:

   ```
   exec sp_addlogin field_user, field_pwd, hq
   ```
Create the publication and subscription

The remaining task is to define the data to be replicated. To do this, you must first create a publication, which defines the available data, and then create a subscription for field_user, which defines the data that user is sharing.

In Adaptive Server Enterprise, they are created with the sp_create_publication procedure, which creates an empty publication, and the sp_add_article procedure, which adds articles to the procedure. Also, each table must be marked for replication before it can be included in a publication.

❖ To create the publication:

1. Create an empty publication:

   ```
   exec sp_create_publication SalesRepData
   go
   ```

2. Mark both the SalesRep table and the Customer table for publication:

   ```
   exec sp_add_remote_table SalesRep
   go
   exec sp_add_remote_table Customer
   go
   ```

3. Add the whole SalesRep table to the SalesRepData publication:
4 Add the Customer table to the SalesRepData publication, using the rep_key column to partition the table. The following statement should be typed all on one line, except for the go:

```
exec sp_add_article SalesRepData, Customer, NULL, 'rep_key'
go
```

Add a subscription

Each user ID that is to receive changes to a publication must have a subscription to that publication. Subscriptions can only be created for a valid remote user. You need to add a subscription to the SalesRepData publication for the remote database user field_user.

❖ To create a subscription:

1 Create a subscription to SalesRepData for field_user, with a subscription value of rep1:

```
exec sp_subscription 'create', SalesRepData, field_user, 'rep1'
go
```

At this stage, the subscription is not started—that is, no data will be exchanged. The subscription is started by the database extraction utility.

Extract the remote database

There are three stages to producing a remote Adaptive Server Anywhere database:

❖ Extract the schema and data into a set of files. You do this using the ssxtract command-line utility.
❖ Create an Adaptive Server Anywhere database.
❖ Load the schema and data into the database.

Extracting the schema and data

With all the information included, the next step is to extract an Adaptive Server Anywhere database for user field_user. The following command line (entered all on one line, from the tutorial directory) carries out this procedure:

```
ssxtract -v -c "eng=server-name;
dbn=hq;uid=sa;pwd=sysadmin" C:\tutorial\field field_user
```

The command-line arguments have the following meaning.
Setting up the consolidated database

- **-v**  Verbose mode. For development work, this provides additional output.
- **-c**  Connection string argument. The connection string is supplied in double quotes following the `-c`.
- **eng=server-name**  Specifies the server to which the extraction utility is to connect.
- **dbn=hq**  Specifies the database on the server to use; in this case `hq`.
- **uid=sa**  The login ID to use to log on to the database.
- **pwd=sysadmin**  The password to use to log on to the database.
- **C:\tutorial\field**  The directory in which to place files holding the data.
- **field_user**  The user ID for which to extract the database.

For more information on extraction utility command-line switches, see "The extraction command-line utility" on page 315.

Running this command produces the following files:

- **Reload script**  The reload script is named `reload.sql`, and is placed in the current directory.
- **Data files**  Files containing data to load into the database. In this case, these files are empty.

You can create an Adaptive Server Anywhere database using the `dbinit` command-line utility. A simple Adaptive Server Anywhere database is a file, unlike Adaptive Server Enterprise databases.

You should create the Adaptive Server Anywhere database so that it is compatible with Adaptive Server Enterprise database behavior, unless you have set options in your Adaptive Server Enterprise server that are different from the default.

**To create a database file named field.db:**

- Enter the following command from the `c:\tutorial\field` directory:
  ```
  dbinit -b -c -k field.db
  ```

The `-b` switch forces use of blank padding in string comparisons. The `-c` switch enforces case sensitivity for string comparisons. The `-k` switch makes the system catalog more compatible with Adaptive Server Enterprise.
Loading the data into the database

You can load the data into the database using the Adaptive Server Anywhere Interactive SQL utility or the rtsql command-line utility. rtsql is an alternative to Interactive SQL for batch processes only, and is provided for the runtime database.

- **To load the data into the database using Interactive SQL:**
  1. Start an Adaptive Server Anywhere server running on the field database:
     
     ```
     dbeng6 field.db
     ```
  2. Connect to the server using the Interactive SQL utility:
     
     ```
     dbisql -c "eng=field;dbn=field;uid=DBA;pwd=SQL"
     ```
     
     The user ID and password must be entered in upper case, as the Adaptive Server Anywhere database was created as case-sensitive.
  3. Load the data using the READ command:
     
     ```
     READ C:\TUTORIAL\RELOAD.SQL
     ```

- **To load the data into the database as a batch process:**
  1. Start an Adaptive Server Anywhere server running on the field database:
     
     ```
     dbeng6 field.db
     ```
  2. Run the script from Interactive SQL:
     
     ```
     dbisql -c "eng=field;dbn=field;uid=DBA;pwd=SQL"
     reload.sql
     ```
     
     The user ID and password must be entered in upper case, as the Adaptive Server Anywhere database was created as case-sensitive.

What next?

The system is now ready for replication.

👋 For the next step, inserting and replicating data, see the section "Start replicating data" on page 88.
Start replicating data

You now have a replication system in place. In this section, data is replicated from the consolidated database to the remote database, and from the remote to the consolidated database.

Enter data at the consolidated database

In this section we enter data into the SalesRep and Customer tables at the consolidated (Adaptive Server Enterprise) database, and replicate this data to the Adaptive Server Anywhere database.

❖ To enter data at the Adaptive Server Enterprise database:

1 Connect to the Adaptive Server Enterprise server from isql:

```
isql -S server-name -U sa -P sysadmin
```

2 Ensure you are using the hq database, and enter a series of rows:

```
use hq
go
insert into SalesRep (rep_key, name) values ('repl', 'Field User')
go
insert into SalesRep (rep_key, name) values ('rep2', 'Another User')
go
insert into Customer (cust_key, name, rep_key) values ('cust1', 'Ocean Sports', 'repl')
go
insert into Customer (cust_key, name, rep_key) values ('cust2', 'Sports Plus', 'rep2')
go
commit
go
```

Ocean Sports is assigned to Field User, and Sports Plus is assigned to Another User. You must commit the changes, as SQL Remote replicates only committed changes.

Having entered the data at the consolidated database, you now need to send the relevant rows to the remote Adaptive Server Anywhere database.
Send data from the consolidated database

To send the rows to the remote database, you must run the Message Agent at the consolidated database. The ssremote program is the Message Agent for Adaptive Server Enterprise.

- To replicate the data from Adaptive Server Enterprise:
  - Enter the following statement (on a single line) at the command line to run the Message Agent against the consolidated database:
    
    ```bash
    ssremote -c "eng=server-name;dbn=hq;uid=sa;pwd=sysadmin"
    ```
  - Click Shutdown on the Message Agent window to stop the Message Agent when the messages have been sent.

Receive data at the remote database

To receive the insert statement at the remote database, you must run the Message Agent, dbremote, at the remote database.

- To receive the data at Adaptive Server Anywhere:
  1. With the database server running, receive the data using the Message Agent for Adaptive Server Anywhere:
    
    ```bash
    dbremote -c "eng=field;dbn=field;uid=DBA;pwd=SQL"
    ```

    For more information on dbremote command line switches, see "The Message Agent" on page 306.
  2. Click Shutdown on the Message Agent window to stop the Message Agent when the messages have been processed.

The Message Agent window displays status information while running. This information can be output to a log file for record keeping in a production setup.

The Message Agent first receives a message from hq, and then sends a message. This return message contains confirmation of successful receipt of the replication update; such confirmations are part of the SQL Remote message tracking system that ensures message delivery even in the event of message system errors.

Verify that the data has arrived

You should now connect to the remote field database using Interactive SQL, and inspect the SalesRep and Customer tables, to see which rows have been received.
To verify that the data has arrived:
1. Connect to the field database using Interactive SQL.
2. Inspect the SalesRep table by typing the following statement:
   ```sql
   SELECT * FROM SalesRep
   ```
   You will see that the SalesRep table contains both rows entered at the consolidated database. This is because the SalesRepData publication included all the data from the SalesRep table.
3. Inspect the Customer table by typing the following statement:
   ```sql
   SELECT * FROM Customer
   ```
   You will see that the Customer table contains only row (Ocean Sports) entered at the consolidated database. This is because the SalesRepData publication included only those customers assigned to the subscribed Sales Rep.

Replicate from the remote database to the consolidated database

You should now try entering data at the remote database and sending it to the consolidated database. Only the outlines are presented here.

To replicate data from the remote database to the consolidated database:
1. Connect to the field database from Interactive SQL.
2. INSERT a row at the remote database. For example
   ```sql
   INSERT INTO Customer (cust_key, name, rep_key)
   VALUES ('cust3', 'North Land Trading', 'repl')
   ```
3. COMMIT the row:
   ```sql
   COMMIT;
   ```
4. With the field.db database running, run dbremote to send the message to the consolidated database.
   ```bash
   dbremote -c "eng=field;dbn=field;uid=DBA;pwd=SQL"
   ```
   (For Windows 3.x, run the dbremotew equivalent.)
5. Run ssremote to receive the message at the consolidated database:
   ```bash
   ssremote -c "eng=server-name;dbn=hq;uid=sar;pwd=sysadmin"
   ```
6. Connect to the consolidated database and display the Customer table. This now has three rows:
Chapter 5  A Tutorial for Adaptive Server Enterprise Users

```
SELECT *
FROM Customer
```

<table>
<thead>
<tr>
<th>cust_key</th>
<th>name</th>
<th>rep_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust1</td>
<td>Ocean Sports</td>
<td>rep1</td>
</tr>
<tr>
<td>cust2</td>
<td>Sports Plus</td>
<td>rep2</td>
</tr>
<tr>
<td>cust3</td>
<td>North Land Trading</td>
<td>rep1</td>
</tr>
</tbody>
</table>

In this simple example, there is no protection against duplicate entries of primary key values. SQL Remote does provide for such protection. For information, see the chapters on SQL Remote Design.
Start replicating data
This part describes replication design issues for SQL Remote
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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design overview</td>
<td>96</td>
</tr>
<tr>
<td>How statements are replicated</td>
<td>100</td>
</tr>
<tr>
<td>Who gets what?</td>
<td>107</td>
</tr>
<tr>
<td>Replication errors and conflicts</td>
<td>109</td>
</tr>
</tbody>
</table>
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.

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 an 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 \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.

<table>
<thead>
<tr>
<th>Character sets in Adaptive Server Anywhere installations</th>
<th>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 &quot;Database Collations and International Languages&quot; on page 289 of the book <em>Adaptive Server Anywhere User's Guide</em>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character sets in Adaptive Server Enterprise installations</td>
<td>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.</td>
</tr>
<tr>
<td>Character sets in mixed installations</td>
<td>The <code>locales.dat</code> 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.</td>
</tr>
<tr>
<td>Adaptive Server Anywhere collation name</td>
<td>Open Client / Open Server name</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>default</td>
<td>cp850</td>
</tr>
<tr>
<td>850</td>
<td>cp850</td>
</tr>
<tr>
<td>437</td>
<td>cp437</td>
</tr>
<tr>
<td>852</td>
<td>cp852</td>
</tr>
<tr>
<td>860</td>
<td>cp860</td>
</tr>
<tr>
<td>437LATIN1</td>
<td>cp437</td>
</tr>
<tr>
<td>437ESP</td>
<td>cp437</td>
</tr>
<tr>
<td>437SVE</td>
<td>cp437</td>
</tr>
<tr>
<td>819CYR</td>
<td>iso_1</td>
</tr>
<tr>
<td>819DAN</td>
<td>iso_1</td>
</tr>
<tr>
<td>819ELL</td>
<td>iso_1</td>
</tr>
<tr>
<td>819ESP</td>
<td>iso_1</td>
</tr>
<tr>
<td>819ISL</td>
<td>iso_1</td>
</tr>
<tr>
<td>819LATIN1</td>
<td>iso_1</td>
</tr>
<tr>
<td>819LATIN2</td>
<td>iso_1</td>
</tr>
<tr>
<td>819NOR</td>
<td>iso_1</td>
</tr>
<tr>
<td>819RUS</td>
<td>iso_1</td>
</tr>
<tr>
<td>819SVE</td>
<td>iso_1</td>
</tr>
<tr>
<td>819TRK</td>
<td>iso_1</td>
</tr>
<tr>
<td>850CYR</td>
<td>cp850</td>
</tr>
<tr>
<td>850DAN</td>
<td>cp850</td>
</tr>
<tr>
<td>850ELL</td>
<td>cp850</td>
</tr>
<tr>
<td>850ESP</td>
<td>cp850</td>
</tr>
<tr>
<td>850ISL</td>
<td>cp850</td>
</tr>
<tr>
<td>850LATIN1</td>
<td>cp850</td>
</tr>
<tr>
<td>850LATIN2</td>
<td>cp850</td>
</tr>
<tr>
<td>850NOR</td>
<td>cp850</td>
</tr>
<tr>
<td>Adaptive Server Anywhere collation name</td>
<td>Open Client / Open Server name</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>850RUS</td>
<td>cp850</td>
</tr>
<tr>
<td>850SVE</td>
<td>cp850</td>
</tr>
<tr>
<td>850TRK</td>
<td>cp850</td>
</tr>
<tr>
<td>852LATIN2</td>
<td>cp852</td>
</tr>
<tr>
<td>852CYR</td>
<td>cp852</td>
</tr>
<tr>
<td>855CYR</td>
<td>cp855</td>
</tr>
<tr>
<td>857TRK</td>
<td>cp857</td>
</tr>
<tr>
<td>860LATIN1</td>
<td>cp860</td>
</tr>
<tr>
<td>866RUS</td>
<td>cp866</td>
</tr>
<tr>
<td>869ELL</td>
<td>cp869</td>
</tr>
<tr>
<td>SJIS</td>
<td>sjis</td>
</tr>
<tr>
<td>SJIS2</td>
<td>sjis</td>
</tr>
<tr>
<td>EUC_JAPAN</td>
<td>eucjis</td>
</tr>
<tr>
<td>EUC_CHINA</td>
<td>eucgb</td>
</tr>
<tr>
<td>EUC_TAIWAN</td>
<td>eucb5</td>
</tr>
<tr>
<td>EUC_KOREA</td>
<td>eucksc</td>
</tr>
<tr>
<td>UTF8</td>
<td>utf8</td>
</tr>
</tbody>
</table>
How statements are replicated

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 SQL Remote replicates SQL statements that modify data.

Only committed transactions are replicated

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.

Primary keys

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.

An UPDATE is not always an UPDATE

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.

The Message Agent is the component that carries out the replication of statements.

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.

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:

<table>
<thead>
<tr>
<th>ID</th>
<th>Rep</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ann</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>Marc</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>Marc</td>
<td>101</td>
</tr>
</tbody>
</table>

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:
How statements are replicated

- **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 in SQL Remote is different for the Adaptive Server Enterprise Message Agent and the Adaptive Server Anywhere Message Agent.

### Trigger replication from Adaptive Server Enterprise

Trigger actions are replicated from Adaptive Server Enterprise, databases to be sure that triggers are not fired when operations are being applied by the Message Agent. The compiled trigger actions from the Adaptive Server Enterprise server do not cause a problem. The FIRE_TRIGGERS database option prevents triggers from being fired. You can set this option for the user ID used by the Message Agent, but be careful not to 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.

### Trigger replication from Adaptive Server Anywhere

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:

- **Conflict resolution trigger actions**  The actions carried out by conflict resolution, or RESOLVE UPDATE, triggers are 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 are replicated, before UPDATE actions.

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.

### An option to replicate trigger actions

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 `dbremote -t` switch.
How statements are replicated

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 -1 command-line option.

   If the maximum blob size is 100 Kb, run the Message Agents with -1 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.
How statements are replicated

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.

Controlling replication of 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.

<table>
<thead>
<tr>
<th>Not the subscriber list</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

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
```

110
where `procedure-name` is the procedure called when a SQL error is detected.

<table>
<thead>
<tr>
<th>Replication error procedure requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>
CHAPTER 7
SQL Remote Design for Adaptive Server Anywhere

About this chapter
This chapter describes how to design a SQL Remote installation when the consolidated database is an Adaptive Server Anywhere database.

Similar material for Adaptive Server Enterprise
Many of the principles of publication design are the same for Adaptive Server Anywhere and Adaptive Server Enterprise, but there are differences in commands and capabilities. There is a large overlap between this chapter and the corresponding chapter for Adaptive Server Enterprise users, "SQL Remote Design for Adaptive Server Enterprise" on page 159.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design overview</td>
<td>114</td>
</tr>
<tr>
<td>Creating publications</td>
<td>115</td>
</tr>
<tr>
<td>Publication design for Adaptive Server Anywhere</td>
<td>122</td>
</tr>
<tr>
<td>Partitioning tables that do not contain the subscription expression</td>
<td>125</td>
</tr>
<tr>
<td>Sharing rows among several subscriptions</td>
<td>133</td>
</tr>
<tr>
<td>Managing conflicts</td>
<td>142</td>
</tr>
<tr>
<td>Ensuring unique primary keys</td>
<td>152</td>
</tr>
<tr>
<td>Creating subscriptions</td>
<td>157</td>
</tr>
</tbody>
</table>
Design overview

Designing a SQL Remote installation includes the following tasks:

- **Designing publications**  The publications determine what information is shared among which databases.
- **Designing subscriptions**  The subscriptions determine what information each user receives.
- **Implementing the design**  Creating publications and subscriptions for all users in the system.

All administration is 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 Anywhere Database Administrator should perform all SQL Remote configuration tasks.
Creating publications

This section describes how to create simple publications consisting of whole tables, or of column-wise subsets of tables.

Simple publications are also discussed in the chapter "A Tutorial for Adaptive Server Anywhere Users" on page 37.

Creating publications for Adaptive Server Anywhere using Sybase Central

You can add a publication to a database from within the SQL Remote folder of Sybase Central.

❖ To create a publication from Sybase Central:

1 Open the SQL Remote folder for your database, which is inside the database container.
2 Click the Publications folder.
3 Double-click Add Publication. The Publication Wizard is displayed.
4 Follow the instructions in the Wizard.

For more information on how to use Sybase Central, see the Sybase Central online Help.

With Sybase Central, you do not need to know the SQL syntax in order to create publications. The remainder of this section discusses different kinds of publication that can be created. It describes the SQL syntax needed for these publications. However, each of the publications can also be created from Sybase Central.

To send SQL statements to an Adaptive Server Anywhere database, you can use the Interactive SQL utility.

Publishing a whole table

The simplest publication you can make consists of a single article, which consists of a single, entire, table.
Creating publications

❖ To create a publication that includes all the rows and columns of a single table:

♦ Execute a create publication statement specifying the table you wish to publish. The syntax is as follows:

```
CREATE PUBLICATION publication_name {
    TABLE table_name
}
```

Example

♦ The following statement creates a publication that publishes the whole customer table:

```
CREATE PUBLICATION pub_customer {
    TABLE customer
}
```

Publishing some of the columns in a table

Partitioning tables

An article that contains only some of the columns of a table can be called a **column-wise partitioning** of the table. An article that contains only some of the rows of a table can be called a **row-wise partitioning** of the table. Articles can be both column-wise and row-wise partitions of a table.

⚠️ Only column-wise partitioning is described in this section. For information on row-wise partitioning of tables, see "Publishing some of the rows from a table" on page 117.

You create a publication that contains all the rows, but only some of the columns, of a table by specifying a list of columns in the CREATE PUBLICATION statement.

⚠️ Any article must conform to the requirements listed in "Conditions for valid articles" on page 123.

❖ To create a publication that includes all the rows and some of the columns of a table:

♦ Enter a CREATE PUBLICATION statement that includes the column names you wish to include. The syntax is as follows:

```
CREATE PUBLICATION publication_name {
    TABLE table_name ( column_name, ... )
}
```

Example

♦ The following statement creates a publication that publishes all rows of the id, company_name, and city columns of the customer table:

```
CREATE PUBLICATION pub_customer {
    TABLE customer ( id,
```
Publishing a set of tables

When publishing a set of tables, a separate article is required for each table in the publication. The following statement creates a publication including all columns and rows in each of a set of tables from the Adaptive Server Anywhere sample database:

```
CREATE PUBLICATION sales {
    TABLE customer,
    TABLE sales_order,
    TABLE sales_order_items,
    TABLE product
}
```

Not all the tables in the database have been published. For example, subscribers receive the `sales_order` table, but not the `employee` table.

**Notes**

- In the sample database, the `sales_order` and `employee` tables are related by a foreign key (`sales_rep` in the `sales_order` table is a foreign key to the `emp_id` column in the `employee` table). Although this could lead to referential integrity problems in the remote database, they are easily avoided by using the database extraction utility.

  - For a discussion of this and other publication design issues, see "Designing to avoid referential integrity errors" on page 149.

- A slightly different publication design (including an article that contains enough of the `employee` table to satisfy the foreign key relationship) would make the publication more robust; for this section we are publishing whole tables only.

Publishing some of the rows from a table

There are two ways of including only some of the rows in a publication:

- **WHERE clause** You can use a WHERE clause to include a subset of rows in an article. All subscribers to the publication containing this article receive the rows that satisfy the WHERE clause.

- **Subscription expression** You can use a subscription expression to include a different set of rows in different subscriptions to publications containing the article.
Creating publications

You can combine a WHERE clause and a subscription expression in an article.

When to use WHERE and subscription expressions

You should use a Subscription expression when different subscribers to a publication are to receive different rows from a table. The Subscription expression is the most powerful method of partitioning tables.

The WHERE clause is used to exclude a set of rows from all subscriptions to a publication.

Publishing a subset of rows using a WHERE clause

The following is a single-article publication sending relevant order information to Samuel Singer, a sales rep:

```
CREATE PUBLICATION pub_orders_samuel_singer {
    TABLE sales_order WHERE sales_rep = 856
}
```

In Sybase Central, the Publication Wizard guides you through creating a WHERE clause for an article.

❖ To create a publication using a WHERE clause:

❖ Enter a CREATE PUBLICATION statement that includes the rows you wish to include in a WHERE clause. The syntax is as follows:

```
CREATE PUBLICATION publication_name {
    TABLE table_name ( column_name, ... )
    WHERE search-condition
}
```

Example

❖ The following statement creates a publication that publishes the id, company_name, city, and state columns of the customer table, for the customers marked as active in the status column.

```
CREATE PUBLICATION pub_customer {
    TABLE customer {
        id,
        company_name,
        city,
        state
    }
    WHERE status = 'active'
}
```

In this case, the status column is not included in the publication. It must therefore have a default value so that inserts at remote databases will not fail at the consolidated database.
Publishing a subset of rows using a subscription expression

In a mobile workforce situation, a sales publication may be wanted where each sales rep subscribes to their own sales orders, enabling them to update their sales orders locally and replicate the sales to the consolidated database.

Using the WHERE clause model, a separate publication for each sales rep would be needed: the following publication is for sales rep Samuel Singer: each of the other sales reps would need a similar publication.

```sql
CREATE PUBLICATION pub_orders_samuel_singer ( TABLE sales_order 
    WHERE sales_rep = 856 
)
```

To address the needs of setups requiring large numbers of different subscriptions, SQL Remote allows a subscription expression to be associated with an article. Subscriptions receive rows depending on the value of a supplied expression.

Benefits of subscription expressions

Publications using a subscription expression are more compact, easier to understand, and provide better performance than maintaining several WHERE clause publications. The database server must add information to the transaction log, and scan the transaction log to send messages, in direct proportion to the number of publications. The subscription expression allows many different subscriptions to be associated with a single publication, whereas the WHERE clause does not.

❖ To create an article using a subscription expression:

❖ Enter a CREATE PUBLICATION statement that includes the expression you wish to use as a match in the subscription expression. The syntax is as follows:

```sql
CREATE PUBLICATION publication_name ( 
    TABLE table_name ( column_name, ... ) 
    SUBSCRIBE BY expression 
)
```

Example

❖ The following statement creates a publication that publishes the id, company_name, city, and state columns of the customer table, and which matches the rows with subscribers according to the value of the state column:

```sql
CREATE PUBLICATION pub_customer ( 
    TABLE customer ( 
        id, 
        company_name, 
        city, 
        state 
    ) 
    SUBSCRIBE BY state 
)
Creating publications

The following statements subscribe two employees to the publication:
Ann Taylor receives the customers in Georgia (GA), and Sam Singer receives the customers in Massachusetts (MA).

```sql
CREATE SUBSCRIPTION
TO pub_customer ('GA')
FOR Ann_Taylor ;

CREATE SUBSCRIPTION
TO pub_customer ('MA')
FOR Sam_Singer
```

Users can subscribe to more than one publication, and can have more than one subscription to a single publication.

For more information

- For information on how to use subqueries in a publication, see "Partitioning tables that do not contain the subscription expression" on page 125.
- For more information on creating subscriptions, see "Creating subscriptions" on page 157.
- In Sybase Central, the Publication Wizard guides you through creating a subscription expression for an article.

Dropping publications

Publications can be dropped using the DROP PUBLICATION statement. The following statement drops the publication named `pub_orders`.

```sql
DROP PUBLICATION pub_orders
```

Dropping a publication has the side effect of dropping all subscriptions to that publication.

Notes on publications

- The different publication types described above can be combined. A single publication can publish a subset of columns from a set of tables, and use both a WHERE clause to select a set of rows to be replicated and a subscription expression to partition rows by subscription.
- DBA authority is required to create publications.
- Publications can be altered and dropped only by the DBA.
♦ Altering publications in a running SQL Remote setup is likely to cause replication errors, and could lead to loss of data in the replication system unless carried out with care.

♦ Views cannot be included in publications.

♦ Stored procedures cannot be included in publications. For a discussion of how SQL Remote replicates procedures and triggers, see "Replication of procedures" on page 102.

♦ For other considerations of referential integrity, see the section "Designing to avoid referential integrity errors" on page 149.
Publication design for Adaptive Server Anywhere

Once you understand how to create simple publications, you must think about proper publication design. Sound design is an important part of building a successful SQL Remote installation. This section helps set out the principles of sound design as they apply to SQL Remote for Adaptive Server Anywhere.

**Similar material for Adaptive Server Enterprise**

Many of the principles of publication design are the same for Adaptive Server Anywhere and Adaptive Server Enterprise, but there are differences in commands and capabilities. There is a large overlap between this section and the corresponding section for Adaptive Server Enterprise users, "Publication design for Adaptive Server Enterprise" on page 166.

**Design issues overview**

<table>
<thead>
<tr>
<th>Design issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each subscription must be a complete relational database</td>
<td>A remote database shares with the consolidated database the information in their subscriptions. The subscription is both a subset of the relational database held at the consolidated site, and also a complete relational database at the remote site. The information in the subscription is therefore subject to the same rules as any other relational database:</td>
</tr>
<tr>
<td>Foreign key relationships must be valid</td>
<td>For every entry in a foreign key, a corresponding primary key entry must exist in the database. The database extraction utility ensures that the CREATE TABLE statements for remote databases do not have foreign keys defined to tables that do not exist remotely.</td>
</tr>
<tr>
<td>Primary key uniqueness must be maintained</td>
<td>There is no way of checking what new rows have been entered at other sites, but not yet replicated. The design must prevent users at different sites adding rows with identical primary key values, as this would lead to conflicts when the rows are replicated to the consolidated database.</td>
</tr>
<tr>
<td>Transaction integrity must be maintained in the absence of locking</td>
<td>The data in the dispersed database (which consists of the consolidated database and all remote databases) must maintain its integrity in the face of updates at all sites, even though there is no system-wide locking mechanism for any particular row.</td>
</tr>
</tbody>
</table>
♦ **Locking conflicts must be prevented or resolved**  In a SQL Remote installation, there is no method for locking rows across all databases to prevent different users from altering the rows at the same time. Such conflicts must be prevented by designing them out of the system or must be resolved in an appropriate manner at the consolidated database.

These key features of relational databases must be incorporated into the design of your publications and subscriptions. This section describes principles and techniques for sound design.

### Conditions for valid articles

All columns in the primary key must be included in the article.

**Supporting INSERTS at remote databases**

For INSERT statements at a remote database to replicate correctly to the consolidated database, you can exclude from an article only columns that can be left out of a valid INSERT statement. These are:

♦ Columns that allow NULL.
♦ Columns that have defaults.

If you exclude any column that does not satisfy one of these requirements, INSERT statements carried out at a remote database will fail when replicated to the consolidated database.

<table>
<thead>
<tr>
<th>ID</th>
<th>Rep</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ann</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>Marc</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>Shih</td>
<td></td>
</tr>
</tbody>
</table>

Insert fails

<table>
<thead>
<tr>
<th>ID</th>
<th>Rep</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ann</td>
</tr>
<tr>
<td>2</td>
<td>Marc</td>
</tr>
<tr>
<td>3</td>
<td>Shih</td>
</tr>
</tbody>
</table>

Insert succeeds
Using BEFORE triggers as an alternative
An exception to this case is when the consolidated database is an
Adaptive Server Anywhere database, and a BEFORE trigger has been
written to maintain the columns that are not included in the INSERT
statement.

Design tips for performance

This section presents a check list for designing high performance SQL
Remote installations.

♦ Keep the number of publications small  In particular, try not to
reference the same table in many different publications.

The work the database server needs to do is proportional to the number
of publications. Keeping the number low and making effective use of
subscriptions lightens the load on the database server.

When operations occur on a table, the database server and the Message
Agent must do some work for each publication that contains the table.
Having one publication for each remote user will drastically increase the
load on the database server. It is much better to have a few publications
that use SUBSCRIBE BY and have subscriptions for each remote user.
The database server does no additional work when more subscriptions
are added for a publication. The Message Agent is designed to work
efficiently with a large number of subscriptions.

♦ Group publications logically  For example, if there is a table that
every remote user requires, such as a price list table, make a separate
publication for that table. Make one publication for each table where the
data can be partitioned by a column value.

♦ Use subscriptions effectively  When remote users receive similar
subsets of the consolidated database, always use publications that
incorporate SUBSCRIBE BY expressions. Do not create a separate
publication for each remote user.

♦ Pay attention to Update Publication Triggers  In particular:
  ♦ Use the NEW / OLD SUBSCRIBE BY syntax.
  ♦ Tune the SELECT statements to ensure they are accessing the
database efficiently.

♦ Monitor the transaction log size  The larger the transaction log, the
longer it takes the Message Agent to scan it. Rename the log regularly
and use the DELETE_OLD_LOGS option.
Partitioning tables that do not contain the subscription expression

In many cases, the rows of a table need to be partitioned even when the subscription expression does not exist in the table.

The Contact example

The Contact database illustrates why and how to partition tables that do not contain the subscription expression.

Example

Here is a simple database that illustrates the problem.

The tables in the database

The three tables are described in more detail as follows:
### Table: SalesRep

All sales representatives that work for the company. The SalesRep table has the following columns:

- **rep_key** An identifier for each sales representative. This is the primary key.
- **name** The name of each sales representative.

The SQL statement creating this table is as follows:

```sql
CREATE TABLE SalesRep (  
  rep_key CHAR(12) NOT NULL,
  name CHAR(40) NOT NULL,
  PRIMARY KEY (rep_key)
)
```

### Table: Customer

All customers that do business with the company. The Customer table includes the following columns:

- **cust_key** An identifier for each customer. This is the primary key.
- **name** The name of each customer.
- **rep_key** An identifier for the sales representative in a sales relationship. This is a foreign key to the SalesRep table.

The SQL statement creating this table is as follows:

```sql
CREATE TABLE Customer (  
  cust_key CHAR(12) NOT NULL,
  name CHAR(40) NOT NULL,
  rep_key CHAR(12) NOT NULL,
  FOREIGN KEY REFERENCES SalesRep,
  PRIMARY KEY (cust_key)
)
Table | Description
--- | ---
Contact | All individual contacts that do business with the company. Each contact belongs to a single customer. The Contact table includes the following columns:

♦ **contact_key** An identifier for each contact. This is the primary key.
♦ **name** The name of each contact.
♦ **cust_key** An identifier for the customer to which the contact belongs. This is a foreign key to the Customer table.

The SQL statement creating this table is:

```sql
CREATE TABLE Contact (
    contact_key CHAR(12) NOT NULL,
    name CHAR(40) NOT NULL,
    cust_key CHAR(12) NOT NULL,
    FOREIGN KEY REFERENCES Customer,
    PRIMARY KEY (contact_key))
```

**Replication goals**

The goals of the design are to provide each sales representative with the following information:

♦ The complete SalesRep table.
♦ Those customers assigned to them, from the Customer table.
♦ Those contacts belonging to the relevant customers, from the Contact table.

**Partitioning the Customer table in the Contact example**

The Customer table can be partitioned using the rep_key value as a subscription expression. A publication that includes the SalesRep and Customer tables would be as follows:

```sql
CREATE PUBLICATION SalesRepData (TABLE SalesRep
    TABLE Customer SUBSCRIBE BY rep_key)
```

127
Partitioning the Contact table in the Contact example

The Contact table must also be partitioned among the sales representatives, but contains no reference to the sales representative rep_key value. How can the Message Agent match a subscription value against rows of this table, when rep_key is not present in the table?

To solve this problem, you can use a subquery in the Contact article that evaluates to the rep_key column of the Customer table. The publication then looks like this:

```
CREATE PUBLICATION SalesRepData {
  TABLE SalesRep
  TABLE Customer
    SUBSCRIBE BY rep_key
  TABLE Contact
    SUBSCRIBE BY (SELECT rep_key
                   FROM Customer
                   WHERE Contactcust_key = Customercust_key )
}
```

The WHERE clause in the subscription expression ensures that the subquery returns only a single value, as only one row in the Customer table has the cust_key value in the current row of the Contact table.

↗ For an Adaptive Server Enterprise consolidated database, the solution is different. For more information, see "Partitioning tables that do not contain the subscription column" on page 168.

Territory realignment in the Contact example

In territory realignment, rows are reassigned among subscribers. In the present case, territory realignment is the reassignment of rows in the Customer table, and by implication also the Contact table, among the Sales Reps.

When a customer is reassigned to a new sales rep, the Customer table is updated. The UPDATE is replicated as an INSERT or a DELETE to the old and new sales representatives, respectively, so that the customer row is properly transferred to the new sales representative.

↗ For information on the way in which Adaptive Server Anywhere and SQL Remote work together to handle this situation, see "Who gets what?" on page 107.
When a customer is reassigned, the Contact table is unaffected. There are no changes to the Contact table, and consequently no entries in the transaction log pertaining to the Contact table. In the absence of this information, SQL Remote cannot reassign the rows of the Contact table along with the Customer.

This failure will cause referential integrity problems: the Contact table at the remote database of the old sales representative contains a cust_key value for which there is no longer a Customer.

Use triggers to maintain Contacts

The solution is to use a trigger containing a special form of UPDATE statement, which does not make any change to the database tables, but which does make an entry in the transaction log. This log entry contains the before and after values of the subscription expression, and so is of the proper form for the Message Agent to replicate the rows properly.

The trigger must be fired BEFORE operations on the row. In this way, the BEFORE value can be evaluated and placed in the log. Also, the trigger must be fired FOR EACH ROW rather than for each statement, and the information provided by the trigger must be the new subscription expression. The Message Agent can use this information to determine which subscribers receive which rows.

Trigger definition

The trigger definition is as follows:

```
CREATE TRIGGER UpdateCustomer
 BEFORE UPDATE ON Customer
 REFERENCING NEW AS NewRow
   OLD AS OldRow
 FOR EACH ROW
 BEGIN
 // determine the new subscription expression
 // for the Customer table
 UPDATE Contact
 PUBLICATION SalesRepData
 OLD SUBSCRIBE BY { OldRow.rep_key }
 NEW SUBSCRIBE BY { NewRow.rep_key }
 WHERE cust_key = NewRow.cust_key;
 END;
```

A special UPDATE statement for publications

The UPDATE statement in this trigger is of the following special form:

```
 UPDATE table-name
 PUBLICATION publication-name
 { SUBSCRIBE BY subscription-expression |
   OLD SUBSCRIBE BY old-subscription-expression
   NEW SUBSCRIBE BY new-subscription-expression }
 WHERE search-condition
```

Here is what the UPDATE statement clauses mean:
Partitioning tables that do not contain the subscription expression

- The *table-name* indicates the table that must be modified at the remote databases.
- The *publication-name* indicates the publication for which subscriptions must be changed.
- The value of *subscription-expression* is used by the Message Agent to determine both new and existing recipients of the rows. Alternatively, you can provide both OLD and NEW subscription expressions.
- The *WHERE* clause specifies which rows are to be transferred between subscribed databases.

**Notes on the trigger**

- If the trigger uses the following syntax:
  ```
  UPDATE table-name
  PUBLICATION pub-name
  SUBSCRIBE BY sub-expression
  WHERE search-condition
  ```
  the trigger must be a BEFORE trigger. In this case, a BEFORE UPDATE trigger. In other contexts, BEFORE DELETE and BEFORE INSERT are necessary.

- If the trigger uses the alternate syntax:
  ```
  UPDATE table-name
  PUBLICATION publication-name
  OLD SUBSCRIBE BY old-subscription-expression
  NEW SUBSCRIBE BY new-subscription-expression }
  WHERE search-condition
  ```
  The trigger can be a BEFORE or AFTER trigger.

- The UPDATE statement lists the publication and table that is affected. The *WHERE* clause in the statement describes the rows that are affected. No changes are made to the data in the table itself by this UPDATE, it makes entries in the transaction log.

- The subscription expression in this example returns a single value. Subqueries returning multiple values can also be used. The value of the subscription expression must the value after the UPDATE.

  In this case, the only subscriber to the row is the new sales representative. In "Sharing rows among several subscriptions" on page 133, we see cases where there are existing as well as new subscribers.

**Information in the transaction log**

Here we describe the information placed in the transaction log. Understanding this helps in designing efficient publications.

- Assume the following data:
  - SalesRep table
<table>
<thead>
<tr>
<th>rep_key</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>rep1</td>
<td>Ann</td>
</tr>
<tr>
<td>rep2</td>
<td>Marc</td>
</tr>
</tbody>
</table>

♦ Customer table

<table>
<thead>
<tr>
<th>cust_key</th>
<th>name</th>
<th>rep_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust1</td>
<td>Sybase</td>
<td>rep1</td>
</tr>
<tr>
<td>cust2</td>
<td>ASA</td>
<td>rep2</td>
</tr>
</tbody>
</table>

♦ Contact table

<table>
<thead>
<tr>
<th>contact_key</th>
<th>name</th>
<th>cust_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>contact1</td>
<td>David</td>
<td>cust1</td>
</tr>
<tr>
<td>contact2</td>
<td>Stefanie</td>
<td>cust2</td>
</tr>
</tbody>
</table>

♦ Now apply the following territory realignment Update statement

```sql
UPDATE Customer
SET rep_key = 'rep2'
WHERE cust_key = 'cust1'
```

The transaction log would contain two entries arising from this statement: one for the BEFORE trigger on the Contact table, and one for the actual UPDATE to the Customer table.

```sql
SalesRepData - Publication Name
rep1 - BEFORE list
rep2 - AFTER list
UPDATE Contact
SET contact_key = 'contact1',
    name = 'David',
    cust_key = 'cust1'
WHERE contact_key = 'contact1'
```

```sql
SalesRepData - Publication Name
rep1 - BEFORE list
rep2 - AFTER list
UPDATE Customer
SET rep_key = 'rep2'
WHERE cust_key = 'cust1'
```

The Message Agent scans the log for these tags. Based on this information it can determine which remote users get an INSERT, UPDATE or DELETE.
In this case, the BEFORE list was rep1 and the AFTER list is rep2. If the before and after list values are different, the rows affected by the UPDATE statement have "moved" from one subscriber value to another. This means the Message Agent will send a DELETE to all remote users who subscribed by the value rep1 for the Customer record cust1 and send an INSERT to all remote users who subscribed by the value rep2.

If the BEFORE and AFTER lists are identical, the remote user already has the row and an UPDATE will be sent.
Sharing rows among several subscriptions

There are cases where a row may need to be included in several subscriptions. For example, we may have a many-to-many relationship. In this section, we use a case study to illustrate how to handle this situation.

The Policy example

The Policy database illustrates why and how to partition tables when there is a many-to-many relationship in the database.

Example database

Here is a simple database that illustrates the problem.

![Database Diagram]

Each sales representative sells to several customers, and some customers deal with more than one sales representative. In this case, the relationship between Customer and SalesRep is thus a many-to-many relationship.

The tables in the database

The three tables are described in more detail as follows:
### Table: SalesRep

All sales representatives that work for the company. The SalesRep table has the following columns:

- **rep_key** An identifier for each sales representative. This is the primary key.
- **name** The name of each sales representative.

The SQL statement creating this table is as follows:

```sql
CREATE TABLE SalesRep (  
  rep_key CHAR(12) NOT NULL,  
  name CHAR(40) NOT NULL,  
  PRIMARY KEY (rep_key)  
);
```

### Table: Customer

All customers that do business with the company. The Customer table includes the following columns:

- **cust_key** A primary key column containing an identifier for each customer.
- **name** A column containing the name of each customer.

The SQL statement creating this table is as follows:

```sql
CREATE TABLE Customer (  
  cust_key CHAR(12) NOT NULL,  
  name CHAR(40) NOT NULL,  
  PRIMARY KEY (cust_key)  
);
```
Table | Description
--- | ---
Policy | A three-column table that maintains the many-to-many relationship between customers and sales representatives. The Policy table has the following columns:
◆ **policy** A primary key column containing an identifier for the sales relationship.
◆ **cust** A column containing an identifier for the customer representative in a sales relationship.
◆ **rep** A column containing an identifier for the sales representative in a sales relationship.

The SQL statement creating this table is as follows.

```sql
CREATE TABLE Policy (  
policy_key CHAR(12) NOT NULL,
cust_key CHAR(12) NOT NULL,
rep_key CHAR(12) NOT NULL,
FOREIGN KEY ( cust_key )
REFERENCES Customer ( cust_key )
FOREIGN KEY ( rep_key )
REFERENCES SalesRep (rep_key )
PRIMARY KEY ( policy_key )  
);```

Replication goals

The goals of the replication design are to provide each sales representative with the following information:

◆ The entire **SalesRep** table.
◆ Those rows from the **Policy** table that include sales relationships involving the sales rep subscribed to the data.
◆ Those rows from the **Customer** table listing customers that deal with the sales rep subscribed to the data.

New problems

The many-to-many relationship between customers and sales representatives introduces new challenges in maintaining a proper sharing of information:

◆ We have a table (in this case the Customer table) that has no reference to the sales representative value that is used in the subscriptions to partition the data.

Again, this problem is addressed by using a subquery in the publication.

◆ Each row in the **Customer** table may be related to many rows in the **SalesRep** table, and shared with many sales representatives databases.
Sharing rows among several subscriptions

Put another way, the rows of the Contact table in "Partitioning tables that do not contain the subscription expression" on page 125 were partitioned into disjoint sets by the publication. In the present example there are overlapping subscriptions.

To meet the replication goals we again need one publication and a set of subscriptions. In this case, we use two triggers to handle the transfer of customers from one sales representative to another.

The publication

A single publication provides the basis for the data sharing:

```sql
CREATE PUBLICATION SalesRepData {
    TABLE SalesRep,
    TABLE Policy SUBSCRIBE BY rep_key,
    TABLE Customer SUBSCRIBE BY {
        SELECT rep_key FROM Policy
        WHERE Policy.cust_key =
            Customer.cust_key
    },
};
```

The subscription statements are exactly as in the previous example.

How the publication works

The publication includes part or all of each of the three tables. To understand how the publication works, it helps to look at each article in turn:

- **SalesRep table**  There are no qualifiers to this article, so the entire SalesRep table is included in the publication.
  ```
  ...  
  TABLE SalesRep,
  ...  
  ```

- **Policy table**  This article uses a subscription expression to specify a column used to partition the data among the sales reps:
  ```
  ...  
  TABLE Policy
  SUBSCRIBE BY rep_key,
  ...  
  ```

  The subscription expression ensures that each sales rep receives only those rows of the table for which the value of the rep_key column matches the value provided in the subscription.

  The Policy table partitioning is disjoint: there are no rows that are shared with more than one subscriber.

- **Customer table**  A subscription expression with a subquery is used to define the partition. The article is defined as follows:
... TABLE Customer SUBSCRIBE BY {
    SELECT rep_key
    FROM Policy
    WHERE Policy.cust_key =
    Customer.cust_key
},
...

The Customer partitioning is non-disjoint: some rows are shared with more than one subscriber.

Multiple-valued subqueries in publications

The subquery in the Customer article returns a single column (rep_key) in its result set, but may return multiple rows, corresponding to all those sales representatives that deal with the particular customer. When a subscription expression has multiple values, the row is replicated to all subscribers whose subscription matches any of the values. It is this ability to have multiple-valued subscription expressions that allows non-disjoint partitionings of a table.

Territory realignment with a many-to-many relationship

The problem of territory realignment (reassigning rows among subscribers) requires special attention, just as in the section "Territory realignment in the Contact example" on page 128.

You need to write triggers to maintain proper data throughout the installation when territory realignment (reassignment of rows among subscribers) is allowed.

How customers are transferred

In this example, we require that a customer transfer be achieved by deleting and inserting rows in the Policy table.

To cancel a sales relationship between a customer and a sales representative, a row in the Policy table is deleted. In this case, the Policy table change is properly replicated to the sales representative, and the row no longer appears in their database. However, no change has been made to the Customer table, and so no changes to the Customer table are replicated to the subscriber.

In the absence of triggers, this would leave the subscriber with incorrect data in their Customer table. The same kind of problem arises when a new row is added to the Policy table.

Using Triggers to solve the problem

The solution is to write triggers that are fired by changes to the Policy table, which include a special syntax of the UPDATE statement. The special UPDATE statement makes no changes to the database tables, but does make an entry in the transaction log that SQL Remote uses to maintain data in subscriber databases.
A BEFORE INSERT trigger

Here is a trigger that tracks INSERTS into the Policy table, and ensures that remote databases contain the proper data.

```sql
CREATE TRIGGER InsPolicy
BEFORE INSERT ON Policy
REFERENCING NEW AS NewRow
FOR EACH ROW
BEGIN
  UPDATE Customer
  PUBLICATION SalesRepData
  SUBSCRIBE BY {
    SELECT rep_key
    FROM Policy
    WHERE cust_key = NewRow.cust_key
    UNION ALL
    SELECT NewRow.rep_key
  }
  WHERE cust_key = NewRow.cust_key;
END;
```

A BEFORE DELETE trigger

Here is a corresponding trigger that tracks Deletes from the Policy table:

```sql
CREATE TRIGGER DelPolicy
BEFORE DELETE ON Policy
REFERENCING OLD AS OldRow
FOR EACH ROW
BEGIN
  UPDATE Customer
  PUBLICATION SalesRepData
  SUBSCRIBE BY {
    SELECT rep_key
    FROM Policy
    WHERE cust_key = OldRow.cust_key
    AND rep_key <> OldRow.rep_key
  }
  WHERE cust_key = OldRow.cust_key;
END;
```

Some of the features of the trigger are the same as in the previous section. The major new features are that the INSERT trigger contains a subquery, and that this subquery can be multi-valued.

Multiple-valued subqueries

The subquery in the BEFORE INSERT trigger is a UNION expression, and can be multi-valued:

```sql
... SELECT rep_key
FROM Policy
WHERE cust_key = NewRow.cust_key
UNION ALL
SELECT NewRow.rep_key
```
The second part of the UNION is the rep_key value for the new sales representative dealing with the customer, taken from the INSERT statement.

The first part of the UNION is the set of existing sales representatives dealing with the customer, taken from the Policy table.

This illustrates the point that the result set of the subscription query must be all those sales representatives receiving the row, not just the new sales representatives.

The subquery in the BEFORE DELETE trigger is multi-valued:

```sql
SELECT rep_key
FROM Policy
WHERE cust_key = OldRow.cust_key
AND rep_key <> OldRow.rep_key
```

The subquery takes rep_key values from the Policy table. The values include the primary key values of all those sales reps who deal with the customer being transferred (WHERE cust_key = OldRow.cust_key), with the exception of the one being deleted (AND rep_key <> OldRow.rep_key).

This again emphasizes that the result set of the subscription query must be all those values matched by sales representatives receiving the row following the DELETE.

**Notes**

- Data in the Customer table is not identified with an individual subscriber (by a primary key value, for example) and is shared among more than one subscriber. This allows the possibility of the data being updated in more than one remote site between replication messages, which could lead to replication conflicts. You can address this issue either by permissions (allowing only certain users the right to update the Customer table, for example) or by adding RESOLVE UPDATE triggers to the database to handle the conflicts programmatically.

- UPDATES on the Policy table have not been described here. They should either be prevented, or a BEFORE UPDATE trigger is required that combines features of the BEFORE INSERT and BEFORE DELETE triggers shown in the example.
Using the Subscribe_by_remote option with many-to-many relationships

When the Subscribe_by_remote option is ON, operations from remote databases on rows with a subscribe by value of NULL or an empty string will assume the remote user is subscribed to the row. By default, the Subscribe_by_remote option is set to ON. In most cases, this setting is the desired setting.

The Subscribe_by_remote option solves a problem that otherwise would arise with some publications, including the Policy example. This section describes the problem, and how the option automatically avoids it.

The publication uses a subquery for the Customer table subscription expression, because each Customer may belong to several Sales Reps:

```sql
CREATE PUBLICATION SalesRepData {
    TABLE SalesRep,
    TABLE Policy SUBSCRIBE BY rep_key,
    TABLE Customer SUBSCRIBE BY {
        SELECT rep_key FROM Policy
        WHERE Policy.cust_key =
        Customer.cust_key
    },
};
```

Marc Dill is a Sales Rep who has just arranged a policy with a new customer. He inserts a new Customer row and also inserts a row in the Policy table to assign the new Customer to himself.

As the INSERT of the Customer row is carried out by the Message Agent at the consolidated database, Adaptive Server Anywhere records the subscription value in the transaction log, at the time of the INSERT.

Later, when the Message Agent scans the log, it builds a list of subscribers from the subscription expression, and Marc Dill is not on the list, as the row in the Policy table assigning the customer to him has not yet been applied. If Subscribe_by_remote were set to OFF, the result would be that the new Customer is sent back to Marc Dill as a DELETE operation.
As long as Subscribe_by_remote is set to ON, the Message Agent assumes the row belongs to the Sales Rep that inserted it, the INSERT is not replicated back to Marc Dill, and the replication system is intact.

If Subscribe_by_remote is set to OFF, you must ensure that the Policy row is inserted before the Customer row, with the referential integrity violation avoided by postponing checking to the end of the transaction.
Managing conflicts

An UPDATE conflict occurs when the following sequence of events takes place:

1. User 1 updates a row at remote site 1.
2. User 2 updates the same row at remote site 2.
3. The update from User 1 is replicated to the consolidated database.
4. The update from User 2 is replicated to the consolidated database.

When the SQL Remote Message Agent replicates UPDATE statements, it does so as a separate UPDATE for each row. Also, the message contains the old row values for comparison. When the update from User 2 arrives at the consolidated database, the values in the row are not those recorded in the message.

Default conflict resolution

By default, the UPDATE still proceeds, so that the User 2 update (the last to reach the consolidated database) becomes the value in the consolidated database, and is replicated to all other databases subscribed to that row.
In general, the default method of conflict resolution is that the most recent operation (in this case that from User 2) succeeds, and no report is made of the conflict. The update from User 1 is lost. SQL Remote also allows custom conflict resolution, using a trigger to resolve conflicts in a way that makes sense for the data being changed.

**Conflict resolution does not apply to primary key updates**

UPDATE conflicts do not apply to primary key updates. You should not update primary keys in a SQL Remote installation. Primary key conflicts must be excluded from the installation by proper design.

This section describes how you can build conflict resolution into your SQL Remote installation at the consolidated database.

### How SQL Remote handles conflicts

#### When a conflict is detected

SQL Remote replication messages include UPDATE statements as a set of single row updates, each with a VERIFY clause that includes values prior to updating.

An UPDATE conflict is detected by the database server as a failure of the VERIFY clause values to match the rows in the database.

Conflicts are detected and resolved by the Message Agent, but only at a consolidated database. When an UPDATE conflict is detected in a message from a remote database, the Message Agent causes the database server to take two actions:

1. Any conflict resolution (RESOLVE UPDATE) triggers are fired.
2. The UPDATE is applied.

UPDATE statements are applied even if the VERIFY clause values do not match, whether or not there is a RESOLVE UPDATE trigger.

Conflict resolution can take several forms. For example:

- In some applications, resolution could mean reporting the conflict into a table.
- You may wish to keep updates made at the consolidated database in preference to those made at remote sites.
- Conflict resolution can be more sophisticated, for example in resolving inventory numbers in the face of goods deliveries and orders.
Managing conflicts

The method of conflict resolution is different at an Adaptive Server Enterprise consolidated database. For more information, see "How SQL Remote handles conflicts" on page 183.

Implementing conflict resolution

This section describes what you need to do to implement custom conflict resolution in SQL Remote for Adaptive Server Anywhere. The concepts are the same in SQL Remote for Adaptive Server Enterprise, but the implementation is different.

SQL Remote allows you to define conflict resolution triggers to handle UPDATE conflicts. Conflict resolution triggers are fired only at a consolidated database, when messages are applied by a remote user. When an UPDATE conflict is detected at a consolidated database, the following sequence of events takes place.

1. Any conflict resolution triggers defined for the operation are fired.
2. The UPDATE takes place.
3. Any actions of the trigger, as well as the UPDATE, are replicated to all remote databases, including the sender of the message that triggered the conflict.

   In general, SQL Remote for Adaptive Server Anywhere does not replicate the actions of triggers: the trigger is assumed to be present at the remote database. Conflict resolution triggers are fired only at consolidated databases, and so their actions are replicated to remote databases.

4. At remote databases, no RESOLVE UPDATE triggers are fired when a message from a consolidated database contains an UPDATE conflict.
5. The UPDATE is carried out at the remote databases.

At the end of the process, the data is consistent throughout the setup.

UPDATE conflicts cannot happen where data is shared for reading, but each row (as identified by its primary key) is updated at only one site. They only occur when data is being updated at more than one site.

Using conflict resolution triggers

This section describes how to use RESOLVE UPDATE, or conflict resolution triggers.
UPDATE statements with a VERIFY clause

Conflict resolution triggers are fired by the failure of values in the VERIFY clause of an UPDATE statement to match the values in the database before the update. An UPDATE statement with a VERIFY clause takes the following form:

```plaintext
UPDATE table-list
SET column-name = expression, ...
[ FROM table-list ]
[ VERIFY (column-name, ...) ]
VALUES ( expression, ... )
[ WHERE search-condition ]
```

The VERIFY clause compares the values of specified columns to a set of expected values, which are the values that were present in the publisher database when the UPDATE statement was applied there.

The verify clause is useful only for single-row updates. However, multi-row update statements entered at a database are replicated as a set of single-row updates by the Message Agent, so this imposes no constraints on client applications.

Conflict resolution trigger syntax

The syntax for a RESOLVE UPDATE trigger is as follows:

```plaintext
CREATE TRIGGER trigger-name
RESOLVE UPDATE
OF column-name ON table-name
[ REFERENCING [ OLD AS old_val ] ]
[ NEW AS new_val ]
[ REMOTE AS remote_val ]
FOR EACH ROW
BEGIN...
END
```

RESOLVE UPDATE triggers fire before each row is updated. The REFERENCING clause allows access to the values in the row of the table to be updated (OLD), to the values the row is to be updated to (NEW) and to the rows that should be present according to the VERIFY clause (REMOTE). Only columns present in the VERIFY clause can be referenced in the REMOTE AS clause; other columns produce a "column not found" error.

Using the VERIFY_ALL_COLUMNS option

The database option VERIFY_ALL_COLUMNS is OFF by default. If it is set to ON, all columns are verified on replicated updates, and a RESOLVE UPDATE trigger is fired whenever any column is different. If it is set to OFF, only those columns that are updated are checked.

Setting this option to ON makes messages bigger, because more information is sent for each UPDATE.

If this option is set at the consolidated database before remote databases are extracted, it will be set at the remote databases also.
Managing conflicts

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

Using the CURRENT REMOTE USER special constant

The CURRENT REMOTE USER special constant holds the user ID of the remote user sending the message. This can be used in RESOLVE UPDATE triggers that place reports of conflicts into a table, to identify the user producing a conflict.

Conflict resolution examples

This section describes some ways of using RESOLVE UPDATE triggers to handle conflicts.

Resolving date conflicts

Suppose a table in a contact management system has a column holding the most recent contact with each customer.

One representative talks with a customer on a Friday, but does not upload his changes to the consolidated database until the next Monday. Meanwhile, a second representative meets the customer on the Saturday, and updates the changes that evening.

There is no conflict when the Saturday UPDATE is replicated to the consolidated database, but when the Monday UPDATE arrives it finds the row already changed.

By default, the Monday UPDATE would proceed, leaving the column with the incorrect information that the most recent contact occurred on Friday.

Update conflicts on this column should be resolved by inserting the most recent date in the row.

Implementing the solution

The following RESOLVE UPDATE trigger chooses the most recent of the two new values and enters it in the database.

```
CREATE TRIGGER contact_date RESOLVE UPDATE
ON contact
REFERENCING OLD AS old_name
NEW AS new_name
FOR EACH ROW
BEGIN
  IF new_name.contact_date <
      old_name.contact_date THEN
    SET new_name.contact_date
      = old_name.contact_date
  END IF
END
```
If the value being updated is later than the value that would replace it, the new value is reset to leave the entry unchanged.

### Resolving inventory conflicts

Consider a warehouse system for a manufacturer of sporting goods. There is a table of product information, with a `quantity` column holding the number of each product left in stock. An update to this column will typically deplete the quantity in stock or, if a new shipment is brought in, add to it.

A sales representative at a remote database enters an order, depleting the stock of small tank top tee shirts by five, from 28 to 23, and enters this in on her database. Meanwhile, before this update is replicated to the consolidated database, a new shipment of tee shirts comes in, and the warehouse enters the shipment, adding 40 to the `quantity` column to make it 68.

![Diagram](diagram.png)

The warehouse entry gets added to the database: the `quantity` column now shows there are 68 small tank-top tee shirts in stock. When the update from the sales representative arrives, it causes a conflict—Adaptive Server Anywhere detects that the update is from 28 to 23, but that the current value of the column is 68.

By default, the most recent UPDATE succeeds, and the inventory level is set to the incorrect value of 23.
Managing conflicts

In this case the conflict should be resolved by summing the changes to the inventory column to produce the final result, so that a final value of 63 is placed into the database.

Implementing the solution

A suitable RESOLVE UPDATE trigger for this situation would add the increments from the two updates. For example:

```sql
CREATE TRIGGER resolve_quantity
RESOLVE UPDATE OF quantity
ON "DBA".product
REFERENCING OLD AS old_name
NEW AS new_name
REMOTE AS remote_name
FOR EACH ROW
BEGIN
    SET new_name.quantity = new_name.quantity
    + old_name.quantity
    - remote_name.quantity
END
```
This trigger adds the difference between the old value in the consolidated database (68) and the old value in the remote database when the original UPDATE was executed (28) to the new value being sent, before the UPDATE is implemented. Thus, new_val.quantity becomes 63 (= 23 + 68 - 28), and this value is entered into the quantity column.

Consistency is maintained at the remote database as follows:
1. The original remote UPDATE changed the value from 28 to 23.
2. The warehouse's entry is replicated to the remote database, but fails as the old value is not what was expected.
3. The changes made by the RESOLVE UPDATE trigger are replicated to the remote database.

### Reporting conflicts

In some cases, you may not want to alter the default way in which SQL Remote resolves conflicts; you may just want to report the conflicts by storing them in a table. In this way, you can look at the conflict table to see what, if any, conflicts have occurred, and if necessary take action to resolve the conflicts.

### Designing to avoid referential integrity errors

The tables in a relational database are related through foreign key references. The referential integrity constraints applied as a consequence of these references ensure that the database remains consistent. If you wish to replicate only a part of a database, there are potential problems with the referential integrity of the replicated database.

**Referential integrity errors stop replication**

If a remote database receives a message that includes a statement that cannot be executed because of referential integrity constraints, no further messages can be applied to the database (because they come after a message that has not yet been applied), including passthrough statements, which would sit in the message queue.

By paying attention to referential integrity issues while designing publications you can avoid these problems. This section describes some of the more common integrity problems and suggests ways to avoid them.

The sales publication described in "Publishing a set of tables" on page 117 includes the sales_order table:

---

Unreplicated referenced table errors

---

149
CREATE PUBLICATION pub_sales {
    TABLE customer,
    TABLE sales_order,
    TABLE sales_order_items,
    TABLE product
}

The sales_order table has a foreign key to the employee table. The id of the sales rep is a foreign key in the sales_order table referencing the primary key of the employee table. However, the employee table is not included in the publication.

If the publication is created in this manner, new sales orders would fail to replicate unless the remote database has the foreign key reference removed from the sales_order table.

If you use the extraction utility to create the remote databases, the foreign key reference is automatically excluded from the remote database, and this problem is avoided. However, there is no constraint in the database to prevent an invalid value from being inserted into the sales_rep_id column of the sales_order table, and if this happens the INSERT will fail at the consolidated database. To avoid this problem, you can include the employee table (or at least its primary key) in the publication.

**Designing triggers to avoid errors**

Actions performed by triggers are not replicated: triggers that exist at one database in a SQL Remote setup are assumed by the replication procedure to exist at other databases in the setup. When an action that fires a trigger at the consolidated database is replicated at the replicate site, the trigger is automatically fired. By default, the database extraction utility extracts the trigger definitions, so that they are in place at the remote database also.

If a publication includes only a subset of a database, a trigger at the consolidated database may refer to tables or rows that are present at the consolidated database, but not at the remote databases. You can design your triggers to avoid such errors by making actions of the trigger conditional using an IF statement. The following list suggests some ways in which triggers can be designed to work on consolidated and remote databases.

- Have actions of the trigger be conditional on the value of CURRENT PUBLISHER. In this case, the trigger would not execute certain actions at the remote database.
- Have actions of the trigger be conditional on the object_id function not returning NULL. The object_id function takes a table or other object as argument, and returns the ID number of that object or NULL if the object does not exist.
♦ Have actions of the trigger be conditional on a SELECT statement which determines if rows exist.

The RESOLVE UPDATE trigger is a special trigger type for the resolution of UPDATE conflicts, and is discussed in the section "Conflict resolution examples" on page 146. The actions of RESOLVE UPDATE triggers are replicated to remote databases, including the database that caused the conflict.
Ensuring unique primary keys

Users at physically distinct sites can each INSERT new rows to a table, so there is an obvious problem ensuring that primary key values are kept unique.

If two users INSERT a row using the same primary key values, the second INSERT to reach a given database in the replication system will fail. As SQL Remote is a replication system for occasionally-connected users, there can be no locking mechanism across all databases in the installation. It is necessary to design your SQL Remote installation so that primary key errors do not occur.

For primary key errors to be designed out of SQL Remote installations; the primary keys of tables that may be modified at more than one site must be guaranteed unique. There are several ways of achieving this goal. This chapter describes a general, economical and reliable method that uses a pool of primary key values for each site in the installation.

The primary key pool is a table that holds a set of primary key values for each database in the SQL Remote installation. Each remote user receives their own set of primary key values. When a remote user inserts a new row into a table, they use a stored procedure to select a valid primary key from the pool. The pool is maintained by periodically running a procedure at the consolidated database that replenishes the supply.

The method is described using a simple example database consisting of sales representatives and their customers. The tables are much simpler than you would use in a real database; this allows us to focus just on those issues important for replication.

The primary key pool

The pool of primary keys is held in a separate table. The following CREATE TABLE statement creates a primary key pool table:

```sql
CREATE TABLE KeyPool (
    table_name VARCHAR(40) NOT NULL,
    value INTEGER NOT NULL,
    location CHAR(12) NOT NULL,
    PRIMARY KEY (table_name, value),
);
```

The columns of this table have the following meanings:
### Column | Description
---|---
table\_name | Holds the names of tables for which primary key pools must be maintained. In our simple example, if new sales representatives were to be added only at the consolidated database, only the Customer table needs a primary key pool and this column is redundant. It is included to show a general solution.
value | Holds a list of primary key values. Each value is unique for each table listed in table\_name.
location | An identifier for the recipient. In some setups, this could be the same as the rep\_key value of the SalesRep table. In other setups, there will be users other than sales representatives and the two identifiers should be distinct.

For performance reasons, you may wish to create an index on the table:

```sql
CREATE INDEX KeyPoolLocation
ON KeyPool (table_name, location, value);
```

### Replicating the primary key pool

You can either incorporate the key pool into an existing publication, or share it as a separate publication. In this example, we create a separate publication for the primary key pool.

🔧 **To replicate the primary key pool:**

1. Create a publication for the primary key pool data.

   ```sql
   CREATE PUBLICATION KeyPoolData {
       TABLE KeyPool SUBSCRIBE BY location
   };
   ```

2. Create subscriptions for each remote database to the KeyPoolData publication.

   ```sql
   CREATE SUBSCRIPTION
   TO KeyPoolData('user1') FOR user1;
   CREATE SUBSCRIPTION
   TO KeyPoolData('user2') FOR user2;
   ...  
   ```

   The subscription argument is the location identifier.
In some circumstances it makes sense to add the KeyPool table to an existing publication and use the same argument to subscribe to each publication. Here we keep the location and rep_key values distinct to provide a more general solution.

**Filling and replenishing the key pool**

Every time a user adds a new customer, their pool of available primary keys is depleted by one. The primary key pool table needs to be periodically replenished at the consolidated database using a procedure such as the following:

```sql
CREATE PROCEDURE ReplenishPool()
BEGIN
    FOR EachTable AS TableCursor
        CURSOR FOR
            SELECT table_name
            AS CurrTable,
            max(value) as MaxValue
        FROM KeyPool
        GROUP BY table_name
        DO
            FOR EachRep AS RepCursor
                CURSOR FOR
                    SELECT location
                    AS CurrRep, count(*) as NumValues
                FROM KeyPool
                WHERE table_name = CurrTable
                GROUP BY location
                DO
                    // make sure there are 100 values.
                    // Fit the top-up value to your
                    // requirements
                    WHILE NumValues < 100 LOOP
                        SET MaxValue = MaxValue + 1;
                        SET NumValues = NumValues + 1;
                        INSERT INTO KeyPool
                        (table_name, location, value)
                        VALUES
                        (CurrTable, CurrRep, MaxValue);
                    END LOOP;
                END FOR;
            END FOR;
        END FOR;
    END;
```

This procedure fills the pool for each user up to 100 values. The value you need depends on how often users are inserting rows into the tables in the database.

The `ReplenishPool` procedure must be run periodically at the consolidated database to refill the pool of primary key values in the `KeyPool` table.
The ReplenishPool procedure requires at least one primary key value to exist for each subscriber, so that it can find the maximum value and add one to generate the next set. To initially fill the pool you can insert a single value for each user, and then call ReplenishPool to fill up the rest. The following example illustrates this for three remote users and a single consolidated user named Office:

```
INSERT INTO KeyPool VALUES( 'Customer', 40, 'user1' );
INSERT INTO KeyPool VALUES( 'Customer', 41, 'user2' );
INSERT INTO KeyPool VALUES( 'Customer', 42, 'user3' );
INSERT INTO KeyPool VALUES( 'Customer', 43, 'Office' );
CALL ReplenishPool();
```

**Cannot use a trigger to replenish the key pool**
You cannot use a trigger to replenish the key pool, as trigger actions are not replicated.

### Adding new customers

When a sales representative wants to add a new customer to the Customer table, the primary key value to be inserted is obtained using a stored procedure. This example shows a stored procedure to supply the primary key value, and also illustrates a stored procedure to carry out the INSERT.

The procedures takes advantage of the fact that the Sales Rep identifier is the CURRENT PUBLISHER of the remote database.

- **NewKey procedure**  The NewKey procedure supplies an integer value from the key pool and deletes the value from the pool.

```
CREATE PROCEDURE NewKey(
    IN @table_name VARCHAR(40),
    OUT @value INTEGER
)
BEGIN
    DECLARE NumValues INTEGER;

    SELECT count(*), min(value)
    INTO NumValues, @value
    FROM KeyPool
    WHERE table_name = @table_name
    AND location = CURRENT PUBLISHER;
    IF NumValues > 1 THEN
        DELETE FROM KeyPool
        WHERE table_name = @table_name
        AND value = @value;
    ELSE
        // Never take the last value, because
        // ReplenishPool will not work.
```
Ensuring unique primary keys

// The key pool should be kept large enough
// that this never happens.
SET @value = NULL;
END IF;
END;

- **NewCustomer procedure**  The NewCustomer procedure inserts a new customer into the table, using the value obtained by NewKey to construct the primary key.

```sql
CREATE PROCEDURE NewCustomer(
    IN customer_name CHAR(40)
) BEGIN
    DECLARE new_cust_key INTEGER;
    CALL NewKey( 'Customer', new_cust_key );
    INSERT INTO Customer ( cust_key, name, location )
    VALUES ( 'Customer' ||
        CONVERT (CHAR(3), new_cust_key),
        customer_name,
        CURRENT PUBLISHER
    );
END;
```

You may want to enhance this procedure by testing the new_cust_key value obtained from NewKey to check that it is not NULL, and preventing the insert if it is NULL.

**Primary key pool summary**

The primary key pool technique requires the following components:

- **Key pool table**  A table to hold valid primary key values for each database in the installation.

- **Replenishment procedure**  A stored procedure keeps the key pool table filled.

- **Sharing of key pools**  Each database in the installation must subscribe to its own set of valid values from the key pool table.

- **Data entry procedures**  New rows are entered using a stored procedure that picks the next valid primary key value from the pool and delete that value from the key pool.
Creating subscriptions

To subscribe to a publication, each subscriber must be granted REMOTE permissions and a subscription must also be created for that user. The details of the subscription are different depending on whether or not the publication uses a subscription expression.

To subscribe a user to a publication, if that publication has no subscription expression, you need the following information:

- **User ID** The user who is being subscribed to the publication. This user must have been granted remote permissions.
- **Publication name** The name of the publication to which the user is being subscribed.

The following statement creates a subscription for a user ID SamS to the `pub_orders samuel singer` publication, which was created using a WHERE clause:

```sql
CREATE SUBSCRIPTION
TO pub_orders_samuel_singer
FOR SamS
```

To subscribe a user to a publication, if that publication does have a subscription expression, you need the following information:

- **User ID** The user who is being subscribed to the publication. This user must have been granted remote permissions.
- **Publication name** The name of the publication to which the user is being subscribed.
- **Subscription value** The value that is to be tested against the subscription expression of the publication. For example, if a publication has the name of a column containing an employee ID as a subscription expression, the value of the employee ID of the subscribing user must be provided in the subscription. The subscription value is always a string.

The following statement creates a subscription for Samuel Singer (user ID SamS, employee ID 856) to the `pub_orders` publication, defined with a subscription expression `sales_rep`, requesting the rows for Samuel Singer's own sales:

```sql
CREATE SUBSCRIPTION
TO pub_orders ( '856' )
FOR SamS
```

In order to receive and apply updates properly, each subscriber needs to have an initial copy of the data. The synchronization process is discussed in "Synchronizing databases" on page 207.
CHAPTER 8

SQL Remote Design for Adaptive Server Enterprise

About this chapter

This chapter describes how to design a SQL Remote installation when the consolidated database is at an Adaptive Server Enterprise server.

Similar material for Adaptive Server Anywhere

Many of the principles of publication design are the same for Adaptive Server Anywhere and Adaptive Server Enterprise, but there are differences in commands and capabilities. There is a large overlap between this chapter and the corresponding chapter for Adaptive Server Anywhere users, "SQL Remote Design for Adaptive Server Anywhere" on page 113.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design overview</td>
<td>160</td>
</tr>
<tr>
<td>Creating publications</td>
<td>161</td>
</tr>
<tr>
<td>Publication design for Adaptive Server Enterprise</td>
<td>166</td>
</tr>
<tr>
<td>Partitioning tables that do not contain the subscription column</td>
<td>168</td>
</tr>
<tr>
<td>Sharing rows among several subscriptions</td>
<td>175</td>
</tr>
<tr>
<td>Managing conflicts</td>
<td>182</td>
</tr>
<tr>
<td>Ensuring unique primary keys</td>
<td>192</td>
</tr>
<tr>
<td>Creating subscriptions</td>
<td>198</td>
</tr>
</tbody>
</table>
Design overview

Designing a SQL Remote installation includes the following tasks:

- **Designing publications**  The publications determine what information is shared among which databases.
- **Designing subscriptions**  The subscriptions determine what information each user receives.
- **Implementing the design**  Creating publications and subscriptions for all users in the system.

All administration is 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 Sybase System Administrator should perform all SQL Remote configuration tasks.

🔗 For more information about the Adaptive Server Enterprise environment, see your Adaptive Server Enterprise documentation.
Chapter 8  SQL Remote Design for Adaptive Server Enterprise

Creating publications

In this section  This section describes how to create simple publications consisting of whole tables, or of column-wise subsets of tables.

Simple publications are also discussed in the chapter "A Tutorial for Adaptive Server Enterprise Users" on page 65.

Creating publications for Adaptive Server Enterprise using Sybase Central

In Sybase Central, you can add a publication to a database from within the SQL Remote folder. The SQL Remote folder is displayed inside a database container.

To create a publication from Sybase Central:

1. Click the Publications folder, which is inside the SQL Remote folder.
2. Double-click Add Publication. The Publication Wizard is displayed.
3. Follow the instructions in the Wizard.

For more information, see the Sybase Central online Help.

With Sybase Central, you do not need to know the SQL syntax in order to create publications. The remainder of this section discusses different kinds of publication that can be created. It describes the SQL syntax needed for these publications. However, each of the publications can also be created from Sybase Central.

Creating whole-table articles

The simplest type of article is one that includes all the rows and columns of a database table.

To create an article that includes all the rows and columns of a table:

1. Mark the table for replication. You do this by executing the sp_add_remote_table procedure:
   
   ```
   sp_add_remote_table table-name
   ```
   
2. Add the table to the publication. You do this by executing the sp_add_article procedure:
Creating publications

sp_add_article publication-name, table-name

Example

♦ The following commands add the table SalesRep to the SalesRepData publication:

sp_add_remote_table 'SalesRep'
sp_add_article 'SalesRepData', 'SalesRep'
go

Creating articles containing some of the columns in a table

To create an article that includes only some of the columns from a table, you need to list the columns that you wish to include, using sp_add_article_col. If no columns are listed, the article includes all columns of the table.

♦ To create an article that includes some of the columns and all the rows of a table:

1 Mark the table for replication. You do this by executing the sp_add_remote_table procedure:

   sp_add_remote_table table-name
   go

2 Add the table to the publication. You do this by executing the sp_add_article procedure:

   sp_add_article publication-name, table-name
   go

   The sp_add_article procedure adds a table to a publication. By default, all columns of the table are added to the publication. If you wish to add only some of the columns, you must use the sp_add_article_col procedure to specify which columns you wish to include.

3 Add individual columns to the publication. You do this by executing the sp_add_article_col procedure for each column:

   sp_add_article_col publication-name, table-name, column-name
   go

Example

♦ The following commands add only the rep_key column of the table SalesRep to the SalesRepData publication:

sp_add_remote_table 'SalesRep'
sp_add_article 'SalesRepData', 'SalesRep'
sp_add_article_col 'SalesRepData', 'SalesRep'
'rep_key'
go

Creating articles containing some of the rows in a table

There are two ways of including only some of the rows from a table in an article:

- **WHERE clause** You can use a WHERE clause to include a subset of rows in an article. All subscribers to the publication containing this article receive the rows that satisfy the WHERE clause.

- **subscription column** You can use a subscription column to include a different set of rows in different subscriptions to publications containing the article.

**Allowed clauses**

In SQL Remote for Adaptive Server Enterprise, the following limitations apply to each of these cases:

- **WHERE clause limitations** The only form of WHERE clause supported is the following:

  ```sql
  WHERE column-name IS NOT NULL.
  ```

- **Subscription column** SQL Remote for Adaptive Server Anywhere supports expressions other than column names. For Adaptive Server Enterprise, the subscription expression must be a column name.

**When to use WHERE and SUBSCRIBE BY**

You should use a subscription expression when different subscribers to a publication are to receive different rows from a table. The subscription expression is the most powerful method of partitioning tables.

Creating an article using a WHERE clause

The WHERE clause is used to exclude a set of rows from all subscriptions to a publication.

- **To create an article using a WHERE clause:**
  1. If you have not already done so, mark the table for replication. You do this by executing the `sp_add_remote_table` procedure:

    ```sql
    sp_add_remote_table table_name
    ```

  2. Add the table to the publication. You do this by executing the `sp_add_article` procedure: Specify the column name corresponding to the **WHERE column IS NOT NULL** clause in the third argument to the procedure:
Creating publications

```
sp_add_article publication_name,
    table_name,
    column_name
```

Do not specify IS NOT NULL; it is implicit. Specify the column name only.

3 If you wish to include only a subset of the columns in the table, specify the columns using the `sp_add_article_col` procedure. You must include the column specified in your WHERE clause in the article.

**Example**

- The following set of statements create a publication containing a single article, which includes only those rows of `test_table` for which column `col_1` is not null:

  ```
  sp_create_publication test_pub
  sp_add_remote_table test_table
  sp_add_article test_pub, test_table, col_1
  go
  ```

Creating an article using a subscription column

The subscription column is used when rows are to be shared among many remote databases.

**To create an article using a subscription column:**

1 If you have not already done so, mark the table for replication. You do this by executing the `sp_add_remote_table` procedure:

   ```
   sp_add_remote_table table_name
   ```

2 Add the table to the publication. You do this by executing the `sp_add_article` procedure: Specify the column name you wish to use as a subscription expression in the fourth argument to the procedure:

   ```
   sp_add_article publication_name, table_name, NULL, column_name
   ```

   You must include the NULL entry to avoid adding a WHERE clause.

3 If you wish to include only a subset of the columns in the table, specify the columns using the `sp_add_article_col` procedure. You must include the column specified in your subscription expression in the article.

**Example**

- The following set of statements create a publication containing a single article, which supports subscriptions based on the value of column `col_1`:

  ```
  sp_create_publication test_pub
  sp_add_remote_table test_table
  sp_add_article test_pub,
  ```
test_table, NULL, col_1
go

Notes on articles

♦ You can combine a WHERE clause and a subscription expression in an article.

♦ All columns in the primary key must be included in any article.

♦ You must not include a subset of columns in an article unless either:
  ♦ The remaining columns have default values or allow NULLs.
  ♦ No inserts are carried out at remote databases. Updates would not cause problems as long as they do not change primary key values.

If you include a subset of columns in an article in situations other than these, INSERT statements at the consolidated database will fail.
Publication design for Adaptive Server Enterprise

Once you understand how to create simple publications, you must think about proper design of publications. This section describes the issues involved in designing publications, and how to take steps towards sound design.

Design issues overview

Each subscription must be a complete relational database

A remote database shares with the consolidated database the information in their subscriptions. The subscription is both a subset of the relational database held at the consolidated site, and also a complete relational database at the remote site. The information in the subscription is therefore subject to the same rules as any other relational database:

♦ **Foreign key relationships must be valid**  For every entry in a foreign key, a corresponding primary key entry must exist in the database.

  The database extraction utility ensures that the CREATE TABLE statements for remote databases do not have foreign keys defined to tables that do not exist remotely.

♦ **Primary key uniqueness must be maintained**  There is no way of checking what new rows have been entered at other sites, but not yet replicated. The design must prevent users at different sites adding rows with identical primary key values, as this would lead to conflicts when the rows are replicated to the consolidated database.

Transaction integrity must be maintained in the absence of locking

The data in the dispersed database (which consists of the consolidated database and all remote databases) must maintain its integrity in the face of updates at all sites, even though there is no system-wide locking mechanism for any particular row.

♦ **Locking conflicts must be prevented or resolved**  In a SQL Remote installation, there is no method for locking rows across all databases to prevent different users from altering the rows at the same time. Such conflicts must be prevented by designing them out of the system or must be resolved in an appropriate manner at the consolidated database.

These key features of relational databases must be incorporated into the design of your publications and subscriptions. This section describes principles and techniques for sound design.
Conditions for valid articles

All columns in the primary key must be included in the article.

Supporting INSERTS at remote databases

For INSERT statements at a remote database to replicate correctly to the consolidated database, you can exclude from an article only columns that can be left out of a valid INSERT statement. These are:

- Columns that allow NULL.
- Columns that have defaults.

If you exclude any column that does not satisfy one of these requirements, INSERT statements carried out at a remote database will fail when replicated to the consolidated database.

<table>
<thead>
<tr>
<th>Consolidated</th>
<th>Insert fails</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO SalesRep (ID, Rep) VALUES (3, 'Shih')</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Rep</td>
</tr>
<tr>
<td>1</td>
<td>Ann</td>
</tr>
<tr>
<td>2</td>
<td>Marc</td>
</tr>
<tr>
<td>3</td>
<td>Shih</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remote</th>
<th>Insert succeeds</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO SalesRep (ID, Rep) VALUES (3, 'Shih')</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Rep</td>
</tr>
<tr>
<td>1</td>
<td>Ann</td>
</tr>
<tr>
<td>2</td>
<td>Marc</td>
</tr>
<tr>
<td>3</td>
<td>Shih</td>
</tr>
</tbody>
</table>

Conditions on rows

There are two ways of including only some of the rows in a publication:

- **WHERE clause** You can use a WHERE clause to include a subset of rows in an article. All subscribers to the publication containing this article receive the rows that satisfy the WHERE clause.

  In SQL Remote for Adaptive Server Enterprise, the only supported WHERE clause is

  WHERE column-name IS NOT NULL

- **Subscription columns** You can use a subscription column to include a different set of rows in different subscriptions to publications containing the article.

  For more information on restrictions on rows, see "Creating articles containing some of the rows in a table" on page 163.
Partitioning tables that do not contain the subscription column

In many cases, the rows of a table need to be partitioned even when the subscription column does not exist in the table. This section describes how to handle this case, using an example.

The Contact example

The Contact database illustrates why and how to partition tables that do not contain the subscription column.

Example

Here is a simple database that illustrates the problem. We call this database the Contact database, because it contains a Contact table in addition to the two tables described earlier in this chapter.

Each sales representative sells to several customers. At some customers there is a single contact, while other customers have several contacts.

The tables in the database

The three tables are described in more detail as follows:
### SalesRep

All sales representatives that work for the company. The SalesRep table has the following columns:

- **rep_key** An identifier for each sales representative. This is the primary key.
- **name** The name of each sales representative.

The SQL statement creating this table is as follows:

```sql
CREATE TABLE SalesRep (  
    rep_key CHAR(12) NOT NULL,  
    name CHAR(40) NOT NULL,  
    PRIMARY KEY (rep_key)  
)  
go
```

### Customer

All customers that do business with the company. The Customer table includes the following columns:

- **cust_key** An identifier for each customer. This is the primary key.
- **name** The name of each customer.
- **rep_key** An identifier for the sales representative in a sales relationship. This is a foreign key to the SalesRep table.

The SQL statement creating this table is as follows:

```sql
CREATE TABLE Customer (  
    cust_key CHAR(12) NOT NULL,  
    name CHAR(40) NOT NULL,  
    rep_key CHAR(12) NOT NULL,  
    FOREIGN KEY (rep_key) REFERENCES SalesRep,  
    PRIMARY KEY (cust_key)  
)  
go
```
Partitioning tables that do not contain the subscription column

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
</table>
| Contact | All individual contacts that do business with the company. Each contact belongs to a single customer. The Contact table includes the following columns:  
  ♦ **contact_key** An identifier for each contact. This is the primary key.  
  ♦ **name** The name of each contact.  
  ♦ **cust_key** An identifier for the customer to which the contact belongs. This is a foreign key to the Customer table.  
  The SQL statement creating this table is:  
  ```sql
  CREATE TABLE Contact (  
    contact_key CHAR(12) NOT NULL,  
    name CHAR(40) NOT NULL,  
    cust_key CHAR(12) NOT NULL,  
    FOREIGN KEY (cust_key)  
    REFERENCES Customer,  
    PRIMARY KEY (contact_key)  
  )
  ```

Replication goals

The goals of the design are to provide each sales representative with the following information:  
♦ The complete SalesRep table.  
♦ Those customers assigned to them, from the Customer table.  
♦ Those contacts belonging to the relevant customers, from the Contact table.  
♦ Maintenance of proper information when Sales Representative territories are realigned.

Territory realignment in the Contact example

In **territory realignment**, rows are reassigned among subscribers. In the current example, territory realignment involves reassigning customers among the sales representatives. It is carried out by updating the **rep_key** column of the **Customer** table.

The UPDATE is replicated as an INSERT or a or a DELETE to the old and new sales representatives, respectively, so that the customer row is properly transferred to the new sales representative.
No log entries for the Contact table when territories realigned

When a customer is reassigned, the Contact table is unaffected. There are no changes to the Contact table, and consequently no entries in the transaction log pertaining to the Contact table. In the absence of this information, SQL Remote cannot reassign the rows of the Contact table along with the Customer. This failure would cause referential integrity problems: the Contact table at the remote database of the old sales representative contains a cust_key value for which there is no longer a Customer.

In this section, we describe how to reassign the rows of the Contact table.

Partitioning the Customer table in the Contact example

The Customer table can be partitioned using the rep_key value as a subscription column. A publication that includes the SalesRep and Customer tables would be as follows:

```sql
exec sp_add_remote_table 'SalesRep'
exec sp_add_remote_table 'Customer'
go
exec sp_create_publication 'SalesRepData'
go
exec sp_add_article 'SalesRepData', 'SalesRep'
exec sp_add_article SalesRepData,
    Customer, NULL,
    'rep_key'
go
```

Adding a subscription-list column to the Contact table

The Contact table must also be partitioned among the sales representatives, but contains no reference to the sales representative rep_key value.

Add a subscription-list column

To solve this problem in Adaptive Server Enterprise, you must add a column to the Contact table containing a comma-separated list of subscription values to the row. (In the present case, there can only be a single subscription value.) The column can be maintained using triggers, so that applications against the database are unaffected by the presence of the column. We call this column a subscription-list column.

When a row in the Customer table is inserted, updated or deleted, a trigger updates rows in the Contact table. In particular, the trigger updates the subscription-list column. As the Contact table is marked for replication, the before and after image of the row is recorded in the log.
Log entries are values, not subscribers

Although in this case the values entered correspond to subscribers, it is not a list of subscribers that is entered in the log. The server handles only information about publications, and the Message Agent handles all information about subscribers. The values entered in the log are for comparison to the subscription value in each subscription. For example, if rows of a table were divided among sales representatives by state or province, the state or province value would be entered in the transaction log.

A subscription-list column is a column added to a table for the sole purpose of holding a comma-separated list of subscribers. In the present case, there can only be a single subscriber to each row of the Contact table, and so the subscription-list column holds only a single value.

🙏 For a discussion of the case where the subscription-list column can hold many values, see "Sharing rows among several subscriptions" on page 175.

In the case of the Contact table, the table definition would be changed to the following:

```sql
CREATE TABLE Contact (  
    contact_key CHAR( 12 ) NOT NULL,  
    name CHAR( 40 ) NOT NULL,  
    cust_key CHAR( 12 ) NOT NULL,  
    subscription_list CHAR( 12 ) NULL,  
    FOREIGN KEY ( cust_key )  
    REFERENCES Customer ( cust_key ),  
    PRIMARY KEY ( contact_key )  
)  
go
```

The additional column is created allowing NULL, so that existing applications can continue to work against the database without change.

The subscription_list column holds the rep_key value corresponding to the row with primary key value cust_key in the Customer table. A set of triggers handles maintenance of the subscription_list column.
\[8\] For an Adaptive Server Anywhere consolidated database, the solution is different. For more information, see "Partitioning tables that do not contain the subscription expression" on page 125.

**Maintaining the subscription-list column**

In order to keep the subscription_list column up to date, triggers are needed for the following operations:

- INSERT on the Contact table.
- UPDATE on the Contact table.
- UPDATE on the Customer table.

The UPDATE of the Customer table addresses the territory realignment problem, where customers are assigned to different Sales Reps.

The trigger for an INSERT on the Contact table sets the subscription_list value to the corresponding rep_key value from the Customer table:

```sql
CREATE TRIGGER set_contact_sub_list
ON Contact
FOR INSERT
AS
BEGIN
    UPDATE Contact
    SET Contact.subscription_list = {
        SELECT rep_key
        FROM Customer
        WHERE Contact.cust_key = Customer.cust_key
    }
    WHERE Contact.contact_key IN {
        SELECT contact_key
        FROM inserted
    }
END
```

An INSERT trigger for the Contact table

Customer

<table>
<thead>
<tr>
<th>cust_key</th>
<th>rep_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust101</td>
<td>rep1</td>
</tr>
<tr>
<td>cust102</td>
<td>rep1</td>
</tr>
<tr>
<td>cust103</td>
<td>rep2</td>
</tr>
<tr>
<td>cust104</td>
<td>rep3</td>
</tr>
</tbody>
</table>

Contact

<table>
<thead>
<tr>
<th>contact_key</th>
<th>cust_key</th>
<th>subscription_list</th>
</tr>
</thead>
<tbody>
<tr>
<td>con1</td>
<td>cust101</td>
<td>rep1</td>
</tr>
<tr>
<td>con2</td>
<td>cust101</td>
<td>rep1</td>
</tr>
<tr>
<td>con3</td>
<td>cust102</td>
<td>rep1</td>
</tr>
<tr>
<td>con4</td>
<td>cust103</td>
<td>rep2</td>
</tr>
<tr>
<td>con5</td>
<td>cust104</td>
<td>rep3</td>
</tr>
</tbody>
</table>
The trigger updates the `subscription_list` column for those rows being inserted; these rows being identified by the subquery

```
SELECT contact_key
FROM inserted
```

An UPDATE trigger for the Contact table

The trigger for an UPDATE on the Contact table checks to see if the `cust_key` column is changed, and if it has updates the `subscription_list` column.

```
CREATE TRIGGER update_contact_sub_list
ON Contact
FOR UPDATE
AS
IF UPDATE ( cust_key )
BEGIN
  UPDATE Contact
  SET subscription_list = Customer.rep_key
  FROM Contact, Customer
  WHERE Contact.cust_key = Customer.cust_key
END
```

The trigger is written using a join; a subquery could also have been used.

An UPDATE trigger for the Customer table

The following trigger handles UPDATES of customers, transferring them to a new Sales Rep:

```
CREATE TRIGGER transfer_contact_with_customer
ON Customer
FOR UPDATE
AS
IF UPDATE ( rep_key )
BEGIN
  UPDATE Contact
  SET Contact.subscription_list = ( 
    SELECT rep_key
    FROM Customer
    WHERE Contact.cust_key = Customer.cust_key 
  )
  WHERE Contact.contact_key IN ( 
    SELECT cust_key
    FROM inserted
  )
END
```
Sharing rows among several subscriptions

There are cases where a row may need to be included in several subscriptions. For example, if instead of the many-to-one relationship between customers and sales representatives that we had above, we may have a many-to-many relationship.

The Policy example

The Policy database illustrates why and how to partition tables when there is a many-to-many relationship in the database.

Example database

Here is a simple database that illustrates the problem.

```
<table>
<thead>
<tr>
<th>Customer</th>
<th>Policy</th>
<th>SalesRep</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_key</td>
<td>policy_key</td>
<td>rep_key</td>
</tr>
<tr>
<td>name</td>
<td>cust_key</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>rep_key</td>
<td></td>
</tr>
</tbody>
</table>
```

The Policy table has a row for each of a set of policies. Each policy is drawn up for a customer by a particular sales representative. There is a many-to-many relationship between customers and sales representatives, and there may be several policies drawn up between a particular rep/customer pair.

Any row in the Customer table may need to be shared with none, one, or several sales representatives.

Solving the problem

To support this case, you need to write triggers to build a comma-delimited list of values to store in a redundant subscription-list column of the Customer table, and include this column as the subscription column when adding the Customer table to the publication. The row is shared with any subscription for which the subscription value matches any of the values in the subscription-list column.

The database, with the subscription-list column included, is as follows:
Sharing rows among several subscriptions

Adaptive Server Enterprise VARCHAR columns are limited to 255 characters, and this limits the number of values that can be stored in the comma-delimited list.

Table definitions

The table definitions are as follows:

```sql
CREATE TABLE SalesRep (  
  rep_key CHAR(12) NOT NULL,  
  name CHAR(40) NOT NULL,  
  PRIMARY KEY (rep_key)  
)  
go
CREATE TABLE Customer (  
  cust_key CHAR(12) NOT NULL,  
  name CHAR(40) NOT NULL,  
  subscription_list VARCHAR(255) NULL,  
  PRIMARY KEY (cust_key)  
)  
go
CREATE TABLE Policy (  
  policy_key INTEGER NOT NULL,  
  cust_key CHAR(12) NOT NULL,  
  rep_key CHAR(12) NOT NULL,  
  FOREIGN KEY (cust_key)  
  REFERENCES Customer(cust_key),  
  FOREIGN KEY (rep_key)  
  REFERENCES SalesRep(rep_key),  
  PRIMARY KEY (policy_key)  
)
```

Notes:

- The `subscription_list` column in the Customer table allows NULLs so that customers can be added who do not have any sales representatives in the `subscription_list` column.

The publication

The publication for this database can be created by the following set of statements:

```sql
//Mark the tables for replication  
exec sp_add_remote_table 'SalesRep'  
exec sp_add_remote_table 'Policy'  
exec sp_add_remote_table 'Customer'
```
go

// Create an empty publication
exec sp_create_publication 'SalesRepData'

// Add the Sales Rep table to the publication
exec sp_add_article 'SalesRepData', 'SalesRep'

// Add the Policy table to the publication
exec sp_add_article 'SalesRepData', 'Policy', NULL, 'rep_key' = -

// Add the Customer table to the publication.
// Subscribe by the subscription_list column
// Exclude the subscription_list column
exec sp_add_article 'SalesRepData', 'Customer', NULL, 'subscription_list'
exec sp_add_article_col 'SalesRepData', 'Customer', 'cust_key'
exec sp_add_article_col 'SalesRepData', 'Customer', 'name'
go

The subscriptions
Subscriptions to this publication take the following form:

exec sp_subscription 'create', 'SalesRepData', 'userID', 'rep_key'
go

where userID identifies the subscriber, and rep_key is the subscription column, which is the value of the rep_key column in the SalesRep table.

Maintaining the subscription-list column

You need to write a procedure and a set of triggers to maintain the subscription-list column added to the Customer table. This section describes these objects.

Stored procedure
The following procedure is used to build the subscription-list column, and is called from the triggers that maintain the subscription_list column.

CREATE PROCEDURE SubscribeCustomer @cust_key CHAR(12)
AS
BEGIN
  -- Rep returns the list of reps for customer @cust_key
  DECLARE Rep CURSOR FOR
    SELECT DISTINCT RTRIM( rep_key )
    FROM Policy
    WHERE cust_key = @cust_key
DECLARE @rep_key CHAR(12)
DECLARE @subscription_list VARCHAR(255)

-- build comma-separated list of rep_key
-- values for this Customer
OPEN Rep
FETCH Rep INTO @rep_key
IF @@sqlstatus = 0 BEGIN
  SELECT @subscription_list = @rep_key
  WHILE 1=1 BEGIN
    FETCH Rep INTO @rep_key
    IF @@sqlstatus != 0 BREAK
    SELECT @subscription_list =
      @subscription_list + ',' + @rep_key
  END
END
ELSE BEGIN
  SELECT @subscription_list = ''
END

-- update the subscription_list in the
-- Customer table
UPDATE Customer
SET subscription_list = @subscription_list
WHERE cust_key = @cust_key
END

Notes:
♦ The procedure takes a Customer key as input argument.
♦ Rep is a cursor for a query that lists each of the Sales Representatives with which the customer has a contract.
♦ The WHILE loop builds a VARCHAR(255) variable holding the comma-separated list of Sales Representatives.
♦ The UPDATE subscription_list column of the Customer

Triggers
The following trigger updates the subscription_list column of the Customer table when a row is inserted into the Policy table.

CREATE TRIGGER InsPolicy
ON Policy
FOR INSERT
AS
BEGIN
  -- Cust returns those customers inserted
  DECLARE Cust CURSOR FOR
    SELECT DISTINCT cust_key
    FROM inserted
  DECLARE @cust_key CHAR(12)

  OPEN Cust
  -- Update the rep list for each Customer
-- with a new rep
WHILE l=1 BEGIN
  FETCH Cust INTO @cust_key
  IF @sqlstatus != 0 BREAK
  EXEC SubscribeCustomer @cust_key
END
END

The following trigger updates the subscription_list column of the Customer table when a row is deleted from the Policy table.

CREATE TRIGGER DelPolicy
ON Policy
FOR DELETE
AS
BEGIN
  -- Cust returns those customers deleted
  DECLARE Cust CURSOR FOR
    SELECT DISTINCT cust_key
    FROM deleted
  DECLARE @cust_key CHAR(12)

  OPEN Cust
  -- Update the rep list for each Customer
  -- losing a rep
  WHILE l=1 BEGIN
    FETCH Cust INTO @cust_key
    IF @sqlstatus != 0 BREAK
    EXEC SubscribeCustomer @cust_key
  END
END

Excluding the subscription-list column from the publication

The subscription-list column should be excluded from the publication, as inclusion of the column leads to excessive updates being replicated.

For example, consider what happens if there are many policies per customer. If a new Sales Representative is assigned to a customer, a trigger fires to update the subscription-list column in the Customer table. If the subscription-list column is part of the publication, then one update for each policy will be replicated to all sales reps that are assigned to this customer.

Triggers at the consolidated database only

The values in the subscription-list column are maintained by triggers. These triggers fire at the consolidated database when the triggering inserts or updates are applied by the Message Agent. The triggers must be excluded from the remote databases, as they maintain a column that does not exist.

You can use the sp_user_extraction_hook procedure to exclude only certain triggers from a remote database on extraction. The procedure is called as the final part of an extraction. By default, it is empty.
Sharing rows among several subscriptions

To customize the extraction procedure to omit certain triggers:

1. Ensure the quoted_identifier option is set to ON:
   ```sql
   set quoted_identifier on
   go
   ```

2. Any temporary tables referenced in the procedure must exist, or the CREATE PROCEDURE statement will fail. The temporary tables referenced in the following procedure are available in the ssremote.sql script. Copy any required table definitions from the script and execute the CREATE TABLE statements, so they exist on the current connection, before creating the procedure.

3. Create the following procedure:
   ```sql
   CREATE PROCEDURE sp_user_extraction_hook
   AS
   BEGIN
   -- We do not want to extract the INSERT and
   -- DELETE triggers created on the Policy table
   -- that maintain the subscription_list
   -- column, since we do not include that
   -- column in the publication.
   -- If these objects were extracted the
   -- INSERTs would fail on the remote database
   -- since they reference a column
   -- ( subscription_list ) that does not exist.
   DELETE FROM #systrigger
   WHERE table_id = object_id('Policy')
   -- Do not create any procedures
   DELETE FROM #sysprocedure
   WHERE proc_name = 'SubscribeCustomer'
   END
   go
   ```

Using the Subscribe_by_remote option with many-to-many relationships

When the Subscribe_by_remote option is ON, operations that arrive from remote databases on rows with a subscribe by value of NULL or " will assume the remote user is subscribed to the row. By default, the Subscribe_by_remote option is set to ON. In most cases, this setting is the desired setting.

The Subscribe_by_remote option solves a problem that otherwise would arise with publications including the Policy example. This section describes how the option automatically avoids the problem.
The database uses a subscription-list column for the Customer table, because each Customer may belong to several Sales Reps:

Marc Dill is a Sales Rep who has just arranged a policy with a new customer. He inserts a new Customer row and also inserts a row in the Policy table to assign the new Customer to himself. Assuming that the subscription-list column is not included in the publication, the operation at Marc's remote database is as follows:

<table>
<thead>
<tr>
<th>Customer</th>
<th>Policy</th>
<th>SalesRep</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust1010</td>
<td>pol2345</td>
<td>195</td>
</tr>
<tr>
<td>cust_name</td>
<td></td>
<td>Marc Dill</td>
</tr>
<tr>
<td>195</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As the INSERT of the Customer row is carried out by the Message Agent at the consolidated database, Adaptive Server Enterprise records the subscription value in the transaction log, at the time of the INSERT.

Later, when the Message Agent scans the log, it builds a list of subscribers to the new row, using the subscription value stored in the log, and Marc Dill is not on that list. If Subscribe_by_remote were set to OFF, the result would be that the new Customer is sent back to Marc Dill as a DELETE operation.

As long as Subscribe_by_remote is set to ON, the Message Agent assumes that, as the subscription-list column is NULL, the row belongs to the Sales Rep that inserted it. As a result, the INSERT is not replicated back to Marc Dill, and the replication system is intact.

You can use a trigger, which executes after the INSERT, to maintain the subscription-list column.
Managing conflicts

An UPDATE conflict occurs when the following sequence of events takes place:

1. User 1 updates a row at remote site 1.
2. User 2 updates the same row at remote site 2.
3. The update from User 1 is replicated to the consolidated database.
4. The update from User 2 is replicated to the consolidated database.

When the SQL Remote Message Agent replicates UPDATE statements, it does so as a separate UPDATE for each row. Also, the message contains the old row values for comparison. When the update from user 2 arrives at the consolidated database, the values in the row are not those recorded in the message.

By default, the UPDATE still proceeds, so that the User 2 update (the last to reach the consolidated database) becomes the value in the consolidated database, and is replicated to all other databases subscribed to that row. In general, the default method of conflict resolution is that the most recent operation (in this case that from User 2) succeeds, and no report is made of the conflict. The update from User 1 is lost.
SQL Remote also allows custom conflict resolution, using a stored procedure to resolve conflicts in a way that makes sense for the data being changed.

UPDATE conflicts do not apply to primary key updates. If the column being updated is a primary key, then when the update from User 2 arrives at the consolidated database, no row will be updated.

This section describes how you can build conflict resolution into your SQL Remote installation at the consolidated database.

### How SQL Remote handles conflicts

#### When a conflict is detected

SQL Remote replication messages include UPDATE statements as a set of single row updates, each including the values prior to updating.

An UPDATE conflict is detected by the database server as a failure of the values to match the rows in the database.

Conflicts are detected and resolved by the Message Agent, but only at a consolidated database. When an UPDATE conflict is detected in a message from a remote database, the Message Agent causes the database server to take two actions:

1. The UPDATE is applied.
2. Any conflict resolution procedures are called.

UPDATE statements are applied even if the VERIFY clause values do not match, whether or not there is a RESOLVE UPDATE trigger.

🔗 The method of conflict resolution is different at an Adaptive Server Anywhere consolidated database. For more information, see "How SQL Remote handles conflicts" on page 143.

### Implementing conflict resolution

This section describes what you need to do to implement custom conflict resolution in SQL Remote.

#### Required objects

For each table on which you wish to resolve conflicts, you must create three database objects to handle the resolution:

- **An old value table** To hold the values that were stored in the table when the conflicting message arrived.
- **A remote value table** To hold the values stored in the table at the remote database when the conflicting update was applied, as determined from the message.
Managing conflicts

- **A stored procedure** To carry out actions to resolve the conflict.

These objects need to exist only in the consolidated database, as that is where conflict resolution occurs. They should not be included in any publications.

**Naming the objects**

When a table is marked for replication, using the `sp_add_remote_table` or `sp_modify_remote_table` stored procedure, optional parameters specify the names of the conflict resolution objects.

The `sp_add_remote_table` and `sp_modify_remote_table` procedures take one compulsory argument, which is the name of the table being marked for replication. It takes three additional arguments, which are the names of the objects used to resolve conflicts. For example, the syntax for `sp_add_remote_table` is:

```sql
exec sp_add_remote_table table_name
[ , resolve_procedure ]
[ , old_row_table ]
[ , remote_row_table ]
```

You must create each of the three objects `resolve_procedure`, `old_row_table`, and `remote_row_table`. These three are discussed in turn.

- **old_row_table** This table must have the same column names and data types as the table `table_name`, but should not have any foreign keys.

  When a conflict occurs, a row is inserted into `old_row_table` containing the values of the row in `table_name` being updated before the UPDATE was applied. Once `resolve_procedure` has been run, the row is deleted.

  As the Message Agent applies updates as a set of single-row updates, the table only ever contains a single row.

- **remote_row_table** This table must have the same column names and data types as the table `table_name`, but should not have any foreign keys.

  When a conflict occurs, a row is inserted into `remote_row_table` containing the values of the row in `table_name` from the remote database before the UPDATE was applied. Once `resolve_procedure` has been run, the row is deleted.

  As the Message Agent applies updates as a set of single-row updates, the table only ever contains a single row.

- **resolve_procedure** This procedure carries out whatever actions are required to resolve a conflict, which may include altering the value in the row or reporting values into a separate table.

Once these objects are created, you must run the `sp_add_remote_table` or `sp_modify_remote_table` procedure to flag them as conflict resolution objects for a table.
Limitations

- At an Adaptive Server Enterprise database, conflict resolution will not work on a table with more than 128 columns while the VERIFY_ALL_COLUMNS option is set to ON. Even if VERIFY_ALL_COLUMNS is set to OFF, if an UPDATE statement updates more than 128 columns, conflict resolution will not work.

A first conflict resolution example

In this example, conflicts in the Customer table in the two-table example used in the tutorials are reported into a table for later review.

The database

The two-table database is as follows:

<table>
<thead>
<tr>
<th>Customer</th>
<th></th>
<th></th>
<th>SalesRep</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust_key</td>
<td>char(12)</td>
<td>rep_key =</td>
<td>rep_key</td>
</tr>
<tr>
<td>name</td>
<td>char(40)</td>
<td>name</td>
<td>char(40)</td>
</tr>
<tr>
<td>rep_key</td>
<td>char(5)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goals of the conflict resolution

The conflict resolution will report conflicts on updates to the name column in the Customer table into a separate table named ConflictLog.

The conflict resolution objects

The conflict resolution tables are defined as follows:

```sql
CREATE TABLE OldCustomer(
  cust_key CHAR(12) NOT NULL,
  name CHAR(40) NOT NULL,
  rep_key CHAR(5) NOT NULL,
  PRIMARY KEY (cust_key)
)

CREATE TABLE RemoteCustomer(
  cust_key CHAR(12) NOT NULL,
  name CHAR(40) NOT NULL,
  rep_key CHAR(5) NOT NULL,
  PRIMARY KEY (cust_key)
)
```

Each of these tables has exactly the same columns and data types as the Customer table itself. The only difference in their definition is that they do not have a foreign key to the SalesRep table.

The conflict resolution procedure reports conflicts into a table named ConflictLog, which has the following definition:

```sql
CREATE TABLE ConflictLog (  
  conflict_key numeric(5,0) identity not null,
  lost_name char(40) not null ,
  won_name char(40) not null ,
  primary key ( conflict_key )
)
The conflict resolution procedure is as follows:

```
CREATE PROCEDURE ResolveCustomer
AS
BEGIN
    DECLARE @cust_key CHAR(12)
    DECLARE @lost_name CHAR(40)
    DECLARE @won_name CHAR(40)

    // Get the name that was lost
    // from OldCustomer
    SELECT @lost_name = name,
           @cust_key = cust_key
    FROM OldCustomer

    // Get the name that won
    // from Customer
    SELECT @won_name = name
    FROM Customer
    WHERE cust_key = @cust_key

    INSERT INTO ConflictLog ( lost_name, won_name )
    VALUES ( @lost_name, @won_name )
END
```

This resolution procedure does not use the RemoteCustomer table.

How the conflict resolution works

The stored procedure is the key to the conflict resolution. It works as follows:

1. Obtains the @lost_name value from the OldCustomer table, and also obtains a primary key value so that the real table can be accessed.

   The @lost_name value is the value that was overridden by the conflict-causing UPDATE.

2. Obtains the @won_name value from the Customer table itself. This is the value that overrode @lost_name. The stored procedure runs after the update has taken place, which is why the value is present in the Customer table. This behavior is different from SQL Remote for Adaptive Server Enterprise, where conflict resolution is implemented in a BEFORE trigger.

3. Adds a row into the ConflictLog table containing the @lost_name and @won_name values.

4. After the procedure is run, the rows in the OldCustomer and RemoteCustomer tables are deleted by the Message Agent. In this simple example, the RemoteCustomer row was not used.
To test the example:

1. Create the tables and the procedure in the consolidated database, and add them as conflict resolution objects to the Customer table.

2. Insert and commit a change at the consolidated database. For example:

   ```
   UPDATE Customer;
   SET name = 'Sea Sports';
   WHERE cust_key='cust1';
   go;
   COMMIT;
   go;
   ```

3. Insert and commit a different change to the same line at the remote database. For example:

   ```
   UPDATE Customer;
   SET name = 'C Sports';
   WHERE cust_key='cust1';
   go;
   COMMIT;
   go;
   ```

4. Replicate the change from the remote to the consolidated database, by running the Message Agent at the remote database to send the message, and then at the consolidated database to receive and apply the message.

5. At the consolidated database, view the `Customer` table and the `ConflictLog` table. The Customer table contains the value from the remote database:

<table>
<thead>
<tr>
<th>cust_key</th>
<th>name</th>
<th>rep_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust1</td>
<td>C Sports</td>
<td>rep1</td>
</tr>
</tbody>
</table>

   The `ConflictLog` table has a single row, showing the conflict:

<table>
<thead>
<tr>
<th>conflict_key</th>
<th>lost_name</th>
<th>won_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sea Sports</td>
<td>C Sports</td>
</tr>
</tbody>
</table>

A second conflict resolution example

This example shows a slightly more elaborate example of resolving a conflict, based on the same situation as the previous example, discussed in "A first conflict resolution example" on page 185.
**Managing conflicts**

**Goals of the conflict resolution**

In this case, the conflict resolution has the following goals:

- Disallow the update from a remote database. The previous example allowed the update.
- Report the name of the remote user whose update failed, along with the lost and won names.

**The conflict resolution objects**

In this case, the **ConflictLog** table has an additional column to record the user ID of the remote user. The table is as follows:

```sql
CREATE TABLE ConflictLog {
    conflict_key numeric(5, 0) identity not null,
    lost_name char(40) not null,
    won_name char(40) not null,
    remote_user char(40) not null,
    primary key (conflict_key)
}
```

The stored procedure is more elaborate. As the update will be disallowed, rather than allowed, the **lost_name** value now refers to the value arriving in the message. It is first applied, but then the conflict resolution procedure replaces it with the value that was previously present.

The stored procedure uses data from the temporary table **#remote**. In order to create a procedure that references a temporary table you first need to create that temporary table. The statement is as follows:

```sql
CREATE TABLE #remote {
    current_remote_user varchar(128),
    current_publisher varchar(128)
}
```

This table is created in TEMPDB, and exists only for the current session. The Message Agent creates its own **#remote** table when it connects, and uses it when the procedure is executed.

```sql
CREATE PROCEDURE ResolveCustomer
AS
BEGIN
    DECLARE @cust_key CHAR(12)
    DECLARE @lost_name CHAR(40)
    DECLARE @won_name CHAR(40)
    DECLARE @remote_user varchar(128)

    -- Get the name that was present before
    -- the message was applied, from OldCustomer
    -- This will "win" in the enx
    SELECT @won_name=name,
           @cust_key=cust_key
    FROM OldCustomer
```
-- Get the name that was applied by the
-- Message Agent from Customer. This will
-- "lose" in the end
SELECT @lost_name=name
FROM Customer
WHERE cust_key = @cust_key

-- Get the remote user value from #remote
SELECT @remote_user = current_remote_user
FROM #remote

-- Report the problem
INSERT INTO ConflictLog ( lost_name, 
        won_name, remote_user )
VALUES ( @lost_name, @won_name, @remote_user )

-- Disallow the update from the Message Agent
-- by resetting the row in the Customer table
UPDATE Customer
SET name = @won_name
WHERE cust_key = @cust_key

END

Notes
There are several points of note here:

✦ The user ID of the remote user is stored by the Message Agent in the
current_remote_user column of the temporary table #remote.

✦ The UPDATE from the Message Agent is applied before the procedure
runs, so the procedure has to explicitly replace the values. This is
different from the case in SQL Remote for Adaptive Server Anywhere,
where conflict resolution is carried out by BEFORE triggers.

Testing the example

✦ To test the example:

1 Create the tables and the procedure in the consolidated database, and add
   them as conflict resolution objects to the Customer table.

2 Insert and commit a change at the consolidated database. For example:

   UPDATE Customer
   SET name = 'Consolidated Sports'
   WHERE cust_key='cust1'
go
   COMMIT
   go

3 Insert and commit a different change to the same line at the remote
database. For example:

   UPDATE Customer
   SET name = 'Field Sports'
WHERE cust_key='cust1'
go
COMMIT

go

4 Replicate the change from the remote to the consolidated database, by running the Message Agent at the remote database to send the message, and then at the consolidated database to receive and apply the message.

5 At the consolidated database, view the **Customer** table and the **ConflictLog** table. The **Customer** table contains the value from the consolidated database:

<table>
<thead>
<tr>
<th>cust_key</th>
<th>name</th>
<th>rep_key</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust1</td>
<td>Consolidated Sports</td>
<td>rep1</td>
</tr>
</tbody>
</table>

The **ConflictLog** table has a single row, showing the conflict and recording the value entered at the remote database:

<table>
<thead>
<tr>
<th>conflict_key</th>
<th>lost_name</th>
<th>won_name</th>
<th>remote_user</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Field Sports</td>
<td>Consolidated Sports</td>
<td>field_user</td>
</tr>
</tbody>
</table>

6 Run the Message Agent again at the remote database. This receives the corrected update from the consolidated database, so that the name of the customer is set to Consolidated Sports here as well.

**Designing to avoid referential integrity errors**

The tables in a relational database are related through foreign key references. The referential integrity constraints applied as a consequence of these references ensure that the database remains consistent. If you wish to replicate only a part of a database, there are potential problems with the referential integrity of the replicated database.

**Referential integrity errors stop replication**

If a remote database receives a message that includes a statement that cannot be executed because of referential integrity constraints, no further messages can be applied to the database (because they come after a message that has not yet been applied), including passthrough statements, which would sit in the message queue.

By paying attention to referential integrity issues while designing publications you can avoid these problems. This section describes some of the more common integrity problems and suggests ways to avoid them.
Consider the following `SalesRepData` publication:

```sql
exec sp_add_remote_table 'SalesRep'
exec sp_create_publication 'SalesRepData'
exec sp_add_article 'SalesRepData', 'SalesRep'
go
```

If the `SalesRep` table had a foreign key to another table (say, `Employee`) that was not included in the publication, inserts or updates to `SalesRep` would fail to replicate unless the remote database had the foreign key reference removed.

If you use the extraction utility to create the remote databases, the foreign key reference is automatically excluded from the remote database, and this problem is avoided. However, there is no constraint in the database to prevent an invalid value from being inserted into the `rep_id` column of the `SalesRep` table, and if this happens the INSERT will fail at the consolidated database. To avoid this problem, you could include the `Employee` table (or at least its primary key) in the publication.
Ensuring unique primary keys

Users at physically distinct sites can each INSERT new rows to a table, so there is an obvious problem ensuring that primary key values are kept unique.

If two users INSERT a row using the same primary key values, the second INSERT to reach a given database in the replication system will fail. As SQL Remote is a replication system for occasionally-connected users, there can be no locking mechanism across all databases in the installation. It is necessary to design your SQL Remote installation so that primary key errors do not occur.

For primary key errors to be designed out of SQL Remote installations; the primary keys of tables that may be modified at more than one site must be guaranteed unique. There are several ways of achieving this goal. This chapter describes a general, economical and reliable method that uses a pool of primary key values for each site in the installation.

The primary key pool is a table that holds a set of primary key values for each database in the SQL Remote installation. Each remote user receives their own set of primary key values. When a remote user inserts a new row into a table, they use a stored procedure to select a valid primary key from the pool. The pool is maintained by periodically running a procedure at the consolidated database that replenishes the supply.

The method is described using a simple example database consisting of sales representatives and their customers. The tables are much simpler than you would use in a real database; this allows us to focus just on those issues important for replication.

The primary key pool

The pool of primary keys is held in a separate table. The following CREATE TABLE statement creates a primary key pool table:

```
CREATE TABLE KeyPool {
    table_name VARCHAR(40) NOT NULL,
    value INTEGER NOT NULL,
    location VARCHAR(6) NOT NULL,
    PRIMARY KEY (table_name, value),
}
```

The columns of this table have the following meanings:
Replicating the primary key pool

You can either incorporate the key pool into an existing publication, or share it as a separate publication. In this example, we create a separate publication for the primary key pool.

To replicate the primary key pool:

1. Create a publication for the primary key pool data.

   ```sql
   sp_create_publication 'KeyPoolData'
go
   sp_add_remote_table 'KeyPool'
go
   sp_add_article 'KeyPoolData', 'KeyPool',
   NULL, 'location'
go
   ```

2. Create subscriptions for each remote database to the `KeyPoolData` publication.

   ```sql
   sp_subscription 'create',
   KeyPoolData,
   field_user,
   repl
   go
   ```

   The subscription argument is the location identifier.
**Ensuring unique primary keys**

In some circumstances it makes sense to add the **KeyPool** table to an existing publication and use the same argument to subscribe to each publication. Here we keep the location and **rep_key** values distinct to provide a more general solution.

### Filling and replenishing the key pool

Every time a user adds a new customer, their pool of available primary keys is depleted by one. The primary key pool table needs to be periodically replenished at the consolidated database using a procedure such as the following:

```sql
CREATE PROCEDURE ReplenishPool AS
BEGIN
    DECLARE @CurrTable VARCHAR(40)
    DECLARE @MaxValue INTEGER
    DECLARE EachTable CURSOR FOR
        SELECT table_name, max(value)
        FROM KeyPool
        GROUP BY table_name
    DECLARE @CurrLoc VARCHAR(6)
    DECLARE @NumValues INTEGER
    DECLARE EachLoc CURSOR FOR
        SELECT location, count(*)
        FROM KeyPool
        WHERE table_name = @CurrTable
        GROUP BY location
    OPEN EachTable
    WHILE 1=1 BEGIN
        FETCH EachTable INTO @CurrTable, @MaxValue
        IF @@sqlstatus != 0 BREAK
        OPEN EachLoc
        WHILE 1=1 BEGIN
            FETCH EachLoc INTO @CurrLoc, @NumValues
            IF @@sqlstatus != 0 BREAK
            -- make sure there are 10 values
            WHILE @NumValues < 10 BEGIN
                SELECT @MaxValue = @MaxValue + 1
                SELECT @NumValues = @NumValues + 1
                INSERT INTO KeyPool
                (table_name, location, value)
                VALUES (@CurrTable, @CurrLoc, @MaxValue)
            END
        END
    CLOSE EachLoc
END
CLOSE EachTable
END
```

194
This procedure fills the pool for each user up to ten values. You may wish to use a larger value in a production environment. The value you need depends on how often users are inserting rows into the tables in the database.

The **ReplenishPool** procedure must be run periodically at the consolidated database to refill the pool of primary key values in the **KeyPool** table.

The **ReplenishPool** procedure requires at least one primary key value to exist for each subscriber, so that it can find the maximum value and add one to generate the next set. To initially fill the pool you can insert a single value for each user, and then call **ReplenishPool** to fill up the rest. The following example illustrates this for three remote users and a single consolidated user named Office:

```sql
INSERT INTO KeyPool VALUES( 'Customer', 40, 'rep1' )
INSERT INTO KeyPool VALUES( 'Customer', 41, 'rep2' )
INSERT INTO KeyPool VALUES( 'Customer', 42, 'rep3' )
INSERT INTO KeyPool VALUES( 'Customer', 43, 'Office' )
EXEC ReplenishPool
```

**Cannot use a trigger to replenish the key pool**

You cannot use a trigger to replenish the key pool, as no actions are replicated to the remote database performing the original operation, including trigger actions.

---

### Adding new customers

When a sales representative wants to add a new customer to the Customer table, the primary key value to be inserted is obtained using a stored procedure. This example shows a stored procedure to supply the primary key value, and also illustrates a stored procedure to carry out the INSERT.

The procedures takes advantage of the fact that the Sales Rep identifier is the CURRENT PUBLISHER of the remote database.

- **NewKey procedure**  The **NewKey** procedure supplies an integer value from the key pool and deletes the value from the pool.

```sql
CREATE PROCEDURE NewKey
    @TableName VARCHAR(40),
    @Location VARCHAR(6),
    @Value INTEGER OUTPUT AS
BEGIN
    DECLARE @NumValues INTEGER
    SELECT @NumValues = count(*),
        @Value = min(value)
    FROM KeyPool
```
WHERE table_name = @TableName
AND location = @Location
IF @NumValues > 1
    DELETE FROM KeyPool
    WHERE table_name = @TableName
    AND value = @Value
ELSE
    -- Never take the last value,
    -- because RestorePool will not work.
    -- The key pool should be kept large
    -- enough so this never happens.
    SELECT @Value = NULL
END

♦ NewCustomer procedure  The NewCustomer procedure inserts a new customer into the table, using the value obtained by NewKey to construct the primary key.

    CREATE PROCEDURE NewCustomer @name VARCHAR(40),
            @loc VARCHAR(6) AS
    BEGIN
        DECLARE @cust INTEGER
        DECLARE @cust_key VARCHAR(12)

        EXEC NewKey 'Customer', @loc, @cust output
        SELECT @cust_key = 'cust' +
                convert( VARCHAR(12), @cust )
        INSERT INTO Customer (cust_key, name, rep_key )
        VALUES ( @cust_key, @name, @loc )
    END

You may want to enhance this procedure by testing the @cust value obtained from NewKey to check that it is not NULL, and preventing the insert if it is NULL.

Testing the key pool

♦ To test the primary key pool:

1  Re-extract a remote database using the field_user user ID.
2  Try this sample INSERT at the remote and consolidated sites:

    EXEC NewCustomer 'Great White North', repl
Primary key pool summary

The primary key pool technique requires the following components:

♦ **Key pool table**  A table to hold valid primary key values for each database in the installation.

♦ **Replenishment procedure**  A stored procedure keeps the key pool table filled.

♦ **Sharing of key pools**  Each database in the installation must subscribe to its own set of valid values from the key pool table.

♦ **Data entry procedures**  New rows are entered using a stored procedure that picks the next valid primary key value from the pool and delete that value from the pool.
Creating subscriptions

To subscribe to a publication, each subscriber must be granted REMOTE permissions and a subscription must also be created for that user. The details of the subscription are different depending on whether or not the publication uses a subscription column.

Subscriptions with no subscription column

To subscribe a user to a publication, if that publication has no subscription column, you need the following information:

♦ **User ID**  The user who is being subscribed to the publication. This user must have been granted remote permissions.

♦ **Publication name**  The name of the publication to which the user is being subscribed.

The following statement creates a subscription for a user ID SamS to the pub_orders_samuel_singer publication, which was created without using a subscription column:

```
sp_subscription 'create',

'pub_orders_samuel_singer',

'SamS'
```

Subscriptions with a subscription column

To subscribe a user to a publication, if that publication does have a subscription column, you need the following information:

♦ **User ID**  The user who is being subscribed to the publication. This user must have been granted remote permissions.

♦ **Publication name**  The name of the publication to which the user is being subscribed.

♦ **Subscription value**  The value that is to be tested against the subscription column of the publication. For example, if a publication has the name of a column containing an employee ID as a subscription column, the value of the employee ID of the subscribing user must be provided in the subscription. The subscription value is always a string.

The following statement creates a subscription for Samuel Singer (user ID SamS, employee ID 856) to the pub_orders publication, defined with a subscription column sales_rep, requesting the rows for Samuel Singer's own sales:

```
sp_subscription create,

'pub_orders',

SamS,

'856'
```
Starting a subscription

In order to receive and apply updates properly, each subscriber needs to have an initial copy of the data. The synchronization process is discussed in "Synchronizing databases" on page 207.
Creating subscriptions
This part describes deployment and administration issues for SQL Remote
CHAPTER 9

Deploying and Synchronizing Databases

About this chapter
This chapter describes the steps you need to take to deploy and synchronize a SQL Remote replication installation.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment overview</td>
<td>204</td>
</tr>
<tr>
<td>Test before deployment</td>
<td>205</td>
</tr>
<tr>
<td>Synchronizing databases</td>
<td>207</td>
</tr>
<tr>
<td>Using the extraction utility</td>
<td>209</td>
</tr>
<tr>
<td>Synchronizing data over a message system</td>
<td>216</td>
</tr>
</tbody>
</table>
Deployment overview

When you have completed the design phase of a SQL Remote system, the next step is to create and deploy the remote databases and applications.

Deployment tasks

In some cases, deployment is a major undertaking. For example, if you have a large number of remote users in a sales force automation system, deployment involves the following steps:

1. Building an Adaptive Server Anywhere database for each remote user, with their own initial copy of the data.

2. Installing the database, together with the Adaptive Server Anywhere database server, the SQL Remote Message Agent, and client application, on each user's machine.

3. Ensuring that the system is properly configured, with correct user names, Message Agent connection strings, permissions, and so on.

In the case of large-scale deployments, remote sites are most commonly Adaptive Server Anywhere databases, and this chapter focuses on this case.

Topics covered

This chapter covers the following topics:

- **Creating remote databases** Before you can deploy a SQL Remote system, you must create a remote database for each remote site.

  Most of the description focuses on creating remote Adaptive Server Anywhere databases.

- **Synchronizing data** Synchronization of a database is the setting up of the initial copy of data in the remote database.
Test before deployment

Thorough testing of your SQL Remote system should be carried out before deployment, especially if you have a large number of remote sites.

When you are in the design and setup phase, you can alter many facets of the SQL Remote setup. Altering publications, message types, writing triggers to resolve update conflicts are all easy to do.

Once you have deployed a SQL Remote application, the situation is different. A SQL Remote setup can be seen as a single dispersed database, spread out over many sites, maintaining a loose form of consistency. The data may never be in exactly the same state in all databases in the setup at once, but all data changes are replicated as complete transactions around the system over time. Consistency is built in to a SQL Remote setup through careful publication design, and through the reconciliation of UPDATE conflicts as they occur.

Upgrading and resynchronization

Once a SQL Remote setup is deployed and is running, it is not easy to tinker with. An upgrade to a SQL Remote installation needs to be carried out with the same care as an initial deployment. This applies also to upgrading maintenance releases of the Adaptive Server Enterprise or Adaptive Server Anywhere database software. Any such software upgrade needs to be tested for compatibility before deployment.

Making changes to a database schema at one database within the system can cause failures because of incompatible database objects. The passthrough mode does allow schema changes to be sent to some or all databases in a SQL Remote setup, but must still be used with care and planning.

The loose consistency in the dispersed database means that updates are always in progress: you cannot generally stop changes being made to all databases, make some changes to the database schema, and restart.

Without careful planning, changes to a database schema will produce errors throughout the installation, and will require all subscriptions to be stopped and resynchronized. Resynchronization involves loading new copies of the data in each remote database, and for more than a few subscribers is a time-consuming process involving work interruptions and possible loss of data.

Changes to avoid on a running system

The following are examples of changes that should not be made to a deployed and running SQL Remote setup. From the list, you will see that there is a class of changes that are permissive, and these are generally permissible, while other changes are restrictive, and must be avoided.

205
The following changes must be avoided, except under the conditions stated:

♦ Change the publisher for the consolidated database.

♦ Make restrictive changes to tables, such as dropping a column or altering a column to not allow NULL values. Changes that include the column or including NULL entries may already be being sent in messages around the SQL Remote setup, and will fail.

♦ Alter a publication. Publication definitions must be maintained at both local and remote sites, and changes that rely on the old publication definition may already be being sent in messages around the SQL Remote setup.

You can make permissive changes, such as adding a new table or column, as long as you use passthrough to ensure that the new table or column exists in the remote database and in the publication at the remote database.

♦ Drop a subscription. This can be done only if you use passthrough deletes to remove the data at the remote site.

♦ Unload and reload an Adaptive Server Anywhere database.

If an Adaptive Server Anywhere database is participating in replication, it cannot be unloaded and reloaded without re-synchronizing the database. Replication is based on the transaction log, and 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.

An Adaptive Server Enterprise database can be unloaded and reloaded as long as the system is quiet and the transaction log is fully scanned. The page_id and row_id rows in the sr_queue_state table of the stable queue must be reset.
Synchronizing databases

What is synchronization?
SQL Remote replication is carried out using the information in the transaction log, but there are two circumstances where SQL Remote deletes all existing rows from those tables of a remote database that form part of a publication, and copies the publication's entire contents from the consolidated database to the remote site. This process is called synchronization.

When to synchronize
Synchronization is used under the following circumstances:

- When a subscription is created at a consolidated database a synchronization is carried out, so that the remote database starts off with a database in the same state as the consolidated database.
- If a remote database gets corrupt or gets out of step with the consolidated database, and cannot be repaired using SQL passthrough mode, synchronization forces the remote site database back in step with the consolidated site.

How to synchronize
Synchronizing a remote database can be done in the following ways:

- **Use the database extraction utility** This utility creates a schema for a remote Adaptive Server Anywhere database, and synchronizes the remote database. This is generally the recommended procedure.
- **Manual synchronization** Synchronize the remote database manually by loading from files, using the PowerBuilder pipeline, or some other tool.
- **Synchronize over the message system** Synchronize the remote database via the message system using the SYNCHRONIZE SUBSCRIPTION statement (Adaptive Server Anywhere) or `sp_subscription 'synchronize'` procedure (Adaptive Server Enterprise).

Caution
Do not execute SYNCHRONIZE SUBSCRIPTION or `sp_subscription 'synchronize'` at a remote database.

Mixed operating systems and database extraction
In many installations, the consolidated server will be running on a different operating system than the remote databases.
Adaptive Server Anywhere databases can be copied from one file or operating system to another. This allows you flexibility in how you carry out your initial synchronization of databases.

Example

For example, you may be running an Adaptive Server Enterprise server on a UNIX system that holds the consolidated database, but wish to deploy remote databases on laptop computers running Windows 95.

In this circumstance, you have several options for the platforms on which you extract the database, including the following, assuming you have the requisite software:

- Run the extraction utility on UNIX to create the reload script and data files. Copy the script and data files to a Windows 95 machine. Create the Adaptive Server Anywhere databases and load them up with the schema and data on Windows 95.
- Run the extraction utility on UNIX to create the reload script and data files. Create the Adaptive Server Anywhere databases and load them up with the schema and data on the same UNIX platform, and then copy the database files onto Windows 95 machines for deployment.
- Run the extraction utility on Windows 95, and carry out all database creation and other tasks on the Windows 95 operating system.

Notes on synchronization and extraction

- Extracting large numbers of subscriptions, or synchronizing subscriptions to large, frequently-used tables, can slow down database access for other users. You may wish to extract such subscriptions when the database is not in heavy use. This happens automatically if you use a SEND AT clause with a quiet time specified.
- Synchronization applies to an entire subscription. There is currently no straightforward way of synchronizing a single table.
- For Adaptive Server Enterprise, Sybase Central does not display subscriptions as started until the Message Agent first runs against the database.
Using the extraction utility

The extraction utility is an aid to creating remote Adaptive Server Anywhere databases. It cannot be used to create remote Adaptive Server Enterprise databases.

Running the extraction utility

The extraction utility can be accessed in the following ways:

♦ From Sybase Central.
♦ As a command-line utility. This is the dbxtract command-line utility (Adaptive Server Anywhere), or the sxtract command-line utility (Adaptive Server Enterprise).

Caution
Do not run the Message Agent while running the extraction utility. The results are unpredictable.

Creating a database from the reload files

The command-line utility unloads a database schema and data suitable for building a remote Adaptive Server Anywhere database for a named subscriber. It produces a SQL command file with default name reload.sql and a set of data files. You can use these files to create a remote Adaptive Server Anywhere database.

Editing of reload.sql may be needed
The database extraction utility is intended to assist in preparing remote databases, but is not intended as a black box solution for all circumstances. You should edit the reload.sql command file as needed when creating remote databases.

❖ To create a remote database from the reload file:

1. Create an Adaptive Server Anywhere database using Sybase Central or using the dbinit utility.
2. Connect to the database from the Interactive SQL utility, and run the reload.sql command file. The following statement entered in the Interactive SQL command window runs the reload.sql command file:

   ```sql
   read path\reload.sql
   ```

   where path is the path of the reload command file.
Using the extraction utility

When used from Sybase Central, the extraction utility carries out the
database unloading task, in the same way that *dbxtract or ssxtract* does, and
then takes the additional step of creating the new database.

The extraction utility does not use a message system. The reload file
(*ssxtract/dbxtract) or database (from Sybase Central) is created in a directory
accessible from the current machine. Synchronizing many subscriptions over
a message link can produce heavy message traffic and, if the message system
is not completely reliable, it may take some time for all the messages to be
properly received at the remote sites.

Before extracting a database

You must complete the following tasks before using the extraction utility at a
consolidated database.

♦ You must have created message types for replication.
♦ You must have added a publisher user ID to the database.
♦ You must have added remote users to the database.
♦ You must have added the publication to the database.
♦ You must have created a subscription for the remote users.

For a description of how to carry out these steps, see the tutorial in the

When you use the extraction utility to create a remote database, the user for
which you are creating the database receives the same permissions they have
in the consolidated database. Further, if the user is a member of any groups
on the consolidated database, those group IDs are created in the remote
database with the permissions they have in the consolidated database.

Using the extraction utility from Sybase Central

For full information on using the extraction utility from Sybase Central, see
the Sybase Central online Help. This section describes one way to extract a
database for a remote user from the current consolidated database.

To extract a database for a remote user:

1 Click the Remote Users folder on the left panel, which is in the SQL
Remote folder. The right panel displays the remote users.
2 Right-click the remote user for whom you wish to extract a database, and select Extract Database from the popup menu. The Extraction Wizard is displayed.

3 Follow the instructions in the Wizard.

For more information

For information about the extraction utility options, available as command-line options or as choices presented by the Database Extraction Wizard, see "Extraction utility options" on page 317.

Designing an efficient extraction procedure

It is very inefficient to create a large number of remote databases by running the extraction utility for each one. You can make the process much more efficient. This section describes one way of making the process more efficient.

There are several potential causes of inefficiency in a large-scale extraction process:

♦ The extraction utility extracts one database at a time, including the schema and data for each user. Commonly, many users share a common schema, and only the data differs. The brute force method of running the extraction utility for each user repeats large amounts of work unnecessarily. Extracting schema and data separately can help with this problem.

♦ Running from Sybase Central, the extraction utility creates a new database for each user. If subscribers share a common schema, you could create a single database, with schema but no data, and copy the file.

♦ By default, the extraction utility runs at isolation level zero. If you are extracting a database from an active server, you should run it at isolation level 3 (see "Extraction utility options" on page 317) to ensure that data in the extracted database is consistent with data on the server.

Running at isolation level 3 may hamper others' turnaround time on the server because of the large number of locks required. It is recommended that you run the extraction utility when the server is not busy, or run it against a copy of the database.

One approach that avoids these problems is as follows:

1 Make a copy of the consolidated database, and at the same time start the subscriptions from the live database. Messages will now start being sent to subscribers, even though they have no database and will not receive them yet.
To start several subscriptions within a single transaction, use the REMOTE RESET statement (Adaptive Server Anywhere) or sp_remote procedure (Adaptive Server Enterprise).

2 Extract the remote databases from the copy of the database. As the database is a copy, there are no locking and concurrency problems. For a large number of remote databases, this process may take several days.

3 As each remote database is created, it is out of date, but its user can receive and apply messages that have been being sent from the live consolidated database, to bring themselves up to date.

This solution interferes with the production database only during the first step. The copy must be made at isolation level three if the database is in use, and uses large numbers of locks. Also, the subscriptions must be started at the same time that the copy is made. Any operations that take place between the copy and the starting of the subscriptions would be lost, and could lead to errors at remote databases.

**Limits to using the extraction utility**

While the extraction utility is the recommended way of creating and synchronizing remote databases from a consolidated databases, there are some circumstances where it cannot be used, and you must synchronize remote databases manually. This section describes some of those cases.

- **Cannot create Adaptive Server Enterprise remote databases** The extraction utility can only be used for Adaptive Server Anywhere remote databases.

- **Additional tables at the remote database** Remote databases can have tables not present at their consolidated database as long as these tables do not take part in replication. Of course, the extraction utility cannot extract such tables from a consolidated database.

- **Adaptive Server Enterprise/Adaptive Server Anywhere differences** Some features in Adaptive Server Enterprise are not present in Adaptive Server Anywhere. The extraction utility carries out a mapping onto similar features, but the mapping is not complete.

  ↗ For more information on Adaptive Server Enterprise/Adaptive Server Anywhere issues, see "Using the extraction utility for Adaptive Server Enterprise" on page 214.
♦ **Extracting procedures and views**  By default, the extraction utility extracts all stored procedures and views from the database. While some of these views and procedures are likely to be required at the remote site, others may not be required—they may refer only to parts of the database that are not included in the remote site.

After running the extraction utility, you should edit the reload script and remove unnecessary views and procedures.

♦ **Using the extraction utility in multi-tiered setups**  To understand the role of the extraction utility in multi-tiered arrangements, consider a three-tiered SQL Remote setup.

This setup is illustrated in the following diagram.

![Diagram of a three-tiered system](image)

From the consolidated database at the top level, you can use the extraction utility to create the second-level databases. You can then add remote users to these second-level databases, and use the extraction utility from each second-level database to create the remote databases. However, if you have to re-extract the second-level databases from the top-level consolidated database, you will delete the remote users that were created, along with their subscriptions and permissions, and will have to rebuild those users. The exception is if you resynchronize data only, in which case you can use the extraction utility to replace the data in the database, without replacing the schema.
Using the extraction utility for Adaptive Server Enterprise

The extraction utility for Adaptive Server Enterprise takes an Adaptive Server Enterprise database schema, and produces an Adaptive Server Anywhere database. There are several limitations and techniques specific to this tool.

Adaptive Server Enterprise features unsupported in Adaptive Server Anywhere

There are some features in Adaptive Server Enterprise that are either not supported or are only partially supported in Adaptive Server Anywhere. The extraction utility handles some of these features partially, and some not at all.

For a full description of Adaptive Server Enterprise/Adaptive Server Anywhere compatibility, see the part Transact-SQL Compatibility, in the Adaptive Server Anywhere User's Guide.

Features not supported in sxsxtract include the following:

- **Grouped procedures**  Adaptive Server Anywhere does not support procedure groups, and they are not extracted by sxsxtract.

- **Named constraints and defaults**  Adaptive Server Anywhere does not support named constraints and named defaults. Any such objects are extracted directly as constraints and defaults that apply to a single object, and the name is lost.

- **Roles**  sxsxtract extracts roles using the Adaptive Server Anywhere concept of groups. It creates a group with the named role, and assigns users to it.

- **Passwords**  If the user for whom a database is being extracted does not have an entry in SYSLOGINS, no password is extracted. If the user does have a login ID, a dummy password is extracted.

- **NCHAR, NVARCHAR**  These data types are extracted as CHAR and VARCHAR, with NULLS allowed.

- **timestamp columns**  Although Adaptive Server Anywhere does provide a timestamp column, it is a different data type from that of Adaptive Server Enterprise. Timestamp columns are not extracted.
Customizing the system tables

The objects that are to be loaded into an Adaptive Server Anywhere database are described in the system catalog. The extraction utility for Adaptive Server Enterprise first creates a set of Adaptive Server Anywhere system tables in TEMPDB, and fills them with data from the Adaptive Server Enterprise catalog. It then unloads this set of tables to provide the reload script that in turn builds an Adaptive Server Anywhere database.

There may be cases where you wish to change the content of the Adaptive Server Anywhere system tables held in TEMPDB. SQL Remote provides a place for you to do that.

The stored procedure that creates and fills the Adaptive Server Anywhere system objects in TEMPDB is called `sp_populate_sql_anywhere`. As its final operation, this procedure calls a procedure called `sp_user_extraction_hook`. This procedure, by default, does nothing. If you wish to customize the extraction procedure, you can do so by writing a suitable `sp_user_extraction_hook` procedure.
## Synchronizing data over a message system

### Creating subscriptions

A subscription is created at a consolidated Adaptive Server Anywhere database using the `sp_subscription` procedure with a first argument of `create`.

Creating a subscription defines the data to be received. It does not synchronize a subscription (provide an initial copy of the data) or start (exchange messages) a subscription.

### Synchronizing subscriptions

Synchronizing a subscription causes the Message Agent to send a copy of all rows in the subscription to the subscriber. It assumes that an appropriate database schema is in place. At an Adaptive Server Anywhere consolidated database, subscriptions are synchronized using the SYNCHRONIZE SUBSCRIPTION statement. At an Adaptive Server Enterprise consolidated database, subscriptions are synchronized using the `sp_subscription` procedure with a first argument of `synchronize`.

When synchronization messages are received at a subscriber database, the Message Agent replaces the current contents of the database with the new copy. Any data at the subscriber that is part of the subscription, and which has not been replicated to the consolidated database, is lost. Once synchronization is complete, the subscription is started by the Message Agent using the START SUBSCRIPTION statement or `sp_subscription` procedure with a first argument of `start`.

**Large volume of messages may result**

Synchronizing databases over a message system may lead to large volumes of messages. In many cases, it is preferable to use the extraction process to synchronize a database locally without placing this burden on the message system.

### Synchronizing subscriptions during operation

If a remote database becomes out of step with the consolidated database, and cannot be brought back in step using the SQL passthrough capabilities of SQL Remote, synchronizing the subscription forces the remote database into step with the consolidated database by copying the rows of the subscription from the consolidated database over the contents at the remote database.

**Data loss on synchronization**

Any data in the remote database that is part of the subscription, but which has not been replicated to the consolidated database, is lost when the subscription is synchronized. You may wish to unload or back up the remote database using Sybase Central or, for Adaptive Server Anywhere, the `dbunload` utility before synchronizing the database.
CHAPTER 10

SQL Remote Administration

About this chapter

This chapter describes general issues and principles for administering a running SQL Remote installation.

For system-specific details, see the chapters "Administering SQL Remote for Adaptive Server Enterprise" on page 277 and "Administering SQL Remote for Adaptive Server Anywhere" on page 255.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management overview</td>
<td>218</td>
</tr>
<tr>
<td>Managing SQL Remote permissions</td>
<td>219</td>
</tr>
<tr>
<td>Using message types</td>
<td>228</td>
</tr>
<tr>
<td>Running the Message Agent</td>
<td>239</td>
</tr>
<tr>
<td>Tuning Message Agent performance</td>
<td>243</td>
</tr>
<tr>
<td>Encoding and compressing messages</td>
<td>250</td>
</tr>
<tr>
<td>The message tracking system</td>
<td>252</td>
</tr>
</tbody>
</table>
Management overview

This chapter describes administration issues for SQL Remote installations. Administration of a deployed and running SQL Remote setup is carried out at a consolidated database.

- **Permissions** As a SQL Remote installation includes many different physical databases, a consistent scheme for users having permissions on remote and consolidated databases is necessary. A section of this chapter describes the considerations you need to make when assigning users permissions.

- **Configuring message systems** Each message system that is used in a SQL Remote installation has control parameters and other settings that must be set up. These settings are discussed in this chapter.

- **The Message Agent** The Message Agent is responsible for sending and receiving messages. While some details of how the Message Agent operates and the configuration options for it, are different for Adaptive Server Anywhere and Adaptive Server Enterprise, some concepts and methods are common to both. These common features are discussed here.

- **Message tracking** Administering a SQL Remote installation means managing large numbers of messages being handed back and forth among many databases. A section on the SQL Remote message tracking system is included to help you understand what the messages contain, when they are sent, how they are applied, and so on.

- **Log management** SQL Remote obtains the data to send from the transaction log. Consequently, proper management of the transaction log, and proper backup procedures, are essential for a smoothly running SQL Remote installation. While many details depend on the server you are running, the generic issues are discussed in this chapter.

- **Passthrough mode** This is a method for directly intervening at a remote site from a consolidated database. This method is discussed in this chapter.
Managing SQL Remote permissions

Users of a database involved in SQL Remote replication are identified by one of the following sets of permissions:

♦ PUBLISH  A single user ID in a database is identified as the publisher for that database. All outgoing SQL Remote messages, including both publication updates and receipt confirmations, are identified by the publisher user ID. Every database in a SQL Remote setup must have a single publisher user ID, as every database in a SQL Remote setup sends messages.

♦ REMOTE  All recipients of messages from the current database, or senders of messages to the current database, who are immediately lower on the SQL Remote hierarchy than the current database must be granted REMOTE permissions.

♦ CONSOLIDATE  At most one user ID may be granted CONSOLIDATE permissions in a database. CONSOLIDATE permissions identifies a database immediately above the current database in a SQL Remote setup. Each database can have only one consolidated database directly above it.

Information about these permissions are held in the SQL Remote system tables, and are independent of other database permissions.

Granting and revoking PUBLISH permissions

When a database sends a message, a user ID representing that database is included with the message to identify its source to the recipient. This user ID is the publisher user ID of the database.

A publisher is required even for read-only remote databases within a replication system, as even these databases send confirmations to the consolidated database to maintain information about the status of the replication. The GRANT PUBLISH statement for remote Adaptive Server Anywhere databases is carried out automatically by the database extraction utility.

Granting and revoking PUBLISH permissions from Sybase Central

You can grant PUBLISH permissions from Sybase Central. You must connect to the database as a user with full system or database administrator permissions.

❖ To set the publisher for a database:

1  Open the SQL Remote folder.
Managing SQL Remote permissions

2 Double click Set Publisher, and select a user from the list.
3 Click OK to set the selected user as the database publisher.

You can also revoke PUBLISH permissions from Sybase Central.

❖ To revoke PUBLISH permissions from Sybase Central:
1 Open the SQL Remote folder. The current publisher is displayed in the right pane.
2 Right-click the current publisher.
3 Click Revoke Publish on the popup menu.

Granting and revoking PUBLISH permissions [Adaptive Server Anywhere]

For Adaptive Server Anywhere, PUBLISH permissions are granted using the GRANT PUBLISH statement:

```
GRANT PUBLISH TO userid ;
```

The `userid` is a user with CONNECT permissions on the current database.
For example, the following statement grants PUBLISH permissions to user `S_Beaulieu`:

```
GRANT PUBLISH TO S_Beaulieu
```

The REVOKE PUBLISH statement revokes the PUBLISH permissions from the current publisher:

```
REVOKE PUBLISH FROM userid
```

Granting and revoking PUBLISH permissions [Adaptive Server Enterprise]

For Adaptive Server Enterprise, PUBLISH permissions are granted using the `sp_publisher` procedure:

```
sp_publisher userid
```

The `userid` is a user with CONNECT permissions on the current database.
For example, the following statement grants PUBLISH permissions to user `S_Beaulieu`:

```
exec sp_publisher 'S_Beaulieu'
go
```

The database is set to have no publisher by executing the `sp_publisher` procedure with no argument:

```
exec sp_publisher

go
```

Notes on PUBLISH permissions

❖ Only one user ID on a database can hold PUBLISH permissions at one time. That user ID is called the **publisher** for the database.

❖ To see the **publisher** user ID in Sybase Central, open the SQL Remote folder; the **publisher** is shown on the right panel.
♦ To see the publisher user ID for an Adaptive Server Anywhere database outside Sybase Central, use the CURRENT PUBLISHER special constant. The following statement retrieves the **publisher** user ID:

```
SELECT CURRENT PUBLISHER
```

♦ To see the publisher user ID for an Adaptive Server Enterprise database outside Sybase Central, use the following statement:

```
SELECT name
FROM sysusers
WHERE uid = ( SELECT user_id
               FROM sr_publisher )
go
```

♦ If PUBLISH permissions is granted to a user ID with GROUP permissions, it is not inherited by members of the group.

♦ PUBLISH permissions carry no authority except to identify the publisher in outgoing messages.

♦ For messages sent from the current database to be received and processed by a recipient, the publisher user ID must have REMOTE or CONSOLIDATE permissions on the receiving database.

♦ The publisher user ID for a database cannot also have REMOTE or CONSOLIDATE permissions on that database. This would identify them as both the sender of outgoing messages and a recipient of such messages.

♦ Changing the user ID of a publisher at a remote database will cause serious problems for any subscriptions that database is involved in, including loss of information. You should not change a remote database publisher user ID unless you are prepared to resynchronize the remote user from scratch.

♦ Changing the user ID of a publisher at a consolidated database while a SQL Remote setup is operating will cause serious problems, including loss of information. You should not change the consolidated database publisher user ID unless you are prepared to close down the SQL Remote setup and resynchronize all remote users.

**Granting and revoking REMOTE and CONSOLIDATE permissions**

REMOTE and CONSOLIDATE permissions are very similar. Each database receiving messages from the current database must have an associated user ID on the current database that is granted one of REMOTE or CONSOLIDATE permissions. This user ID represents the receiving database in the current database.
Managing SQL Remote permissions

Setting REMOTE and CONSOLIDATE permissions

Databases directly below the current database on a SQL Remote hierarchy are granted REMOTE permissions, and the at most one database above the current database in the hierarchy is granted CONSOLIDATE permissions.

For Adaptive Server Anywhere, the GRANT REMOTE and GRANT CONSOLIDATE statements identify the message system and address to which replication messages must be sent.

For Adaptive Server Enterprise, the `sp_grant_remote` procedure sets REMOTE permissions, and the `sp_grant_consolidate` procedure sets CONSOLIDATE permissions.

CONSOLIDATE permissions must be granted even from read-only remote databases to the consolidated database, as receipt confirmations are sent back from the remote databases to the consolidated database. The GRANT CONSOLIDATE statement at remote Adaptive Server Anywhere databases is executed automatically by the database extraction utility.

Granting REMOTE permissions

Each remote database must be represented by a single user ID in the consolidated database. This user ID must be granted REMOTE permissions to identify their user ID and address as a subscriber to publications.

Granting REMOTE permissions accomplishes several tasks:

- It identifies a user ID as a remote user.
- It specifies a message type to use for exchanging messages with this user ID.
- It provides an address to where messages are to be sent.
- It indicates how often messages should be sent to the remote user.

Granting REMOTE permissions is also referred to as adding a remote user to the database.

Sybase Central example

You can add a remote user to a database using Sybase Central.

- **To add a new user to the database, as a remote user:**
  1. Open the Remote Users folder in the SQL Remote folder.
  2. Double-click Add Remote User, and follow the instructions in the wizard.

- **To make an existing user a remote user:**
  1. Open the SQL Remote folder, so the Remote Users folder is displayed in the left pane.
2. Open the Users and Groups folder, so that the user you want to make a remote user is displayed in the right pane.

3. Drag the user icon in the right pane to the Remote Users folder in the SQL Remote folder on the left pane. The Make User a Remote User window is displayed.

4. Select the message type from the list, enter an address, choose the frequency of sending messages, and click OK to make the user a remote user.

**Adaptive Server Anywhere example**

The following statement grants remote permissions to user **S_Beaulieu**, with the following options:

- Use an SMTP e-mail system
- Send messages to e-mail address **s_beaulieu@acme.com**:
- Send message daily, at 10 pm.

```sql
GRANT REMOTE TO S_Beaulieu
TYPE smtp
ADDRESS 's_beaulieu@acme.com'
SEND AT '22:00'
```

**Adaptive Server Enterprise example**

The following statement grants remote permissions to user **S_Beaulieu** with the following options:

- Use the file-sharing system to exchange messages.
- Place messages in the directory **beaulieu** under the address root directory.

The address root directory (for both Adaptive Server Anywhere and Adaptive Server Enterprise) is indicated by the SQLREMOTE environment variable, if it is set. Alternatively, it is indicated by the Directory setting in the FILE message control parameters (held in the registry or INI file).

- Send messages every twelve hours:

```sql
exec sp_grant_remote 'S_Beaulieu',
    'file',
    'beaulieu',
    'SEND EVERY',
    '12:00'
go
```

**Selecting a send frequency**

There are three alternatives for the setting the frequency with which messages are sent. The three alternatives are:
Managing SQL Remote permissions

◇ **SEND EVERY**  A frequency can be specified in hours and minutes.  

When any user with SEND EVERY set is sent messages, all users with the same frequency are sent messages also. For example, all remote users who receive updates every twelve hours are sent updates at the same times, rather than being staggered. This reduces the number of times the Adaptive Server Anywhere transaction log or Adaptive Server Enterprise stable queue has to be processed. You should use as few unique frequencies as possible.

SQL Remote is not intended for up-to-the-minute replication. Frequencies of less than ten minutes are not recommended.

◇ **SEND AT**  A time of day, in hours and minutes.

Updates are started daily at the specified time. It is more efficient to use as few distinct times as possible than to stagger the sending times. Also, choosing times when the database is not busy minimizes interference with other users.

◇ **Default setting (no SEND clause)**  If any user has no SEND AT or SEND EVERY clause, the Message Agent sends messages every time it is run, and then stops: it runs in batch mode.

**Granting CONSOLIDATE permissions**

In the remote database, the publish and subscribe user IDs are inverted compared to the consolidated database. The subscriber (remote user) in the consolidated database becomes the publisher in the remote database. The publisher of the consolidated database becomes a subscriber to publications from the remote database, and is granted CONSOLIDATE permissions.

At each remote database, the consolidated database must be granted CONSOLIDATE permissions. When you produce a remote database by running the database extraction utility, the GRANT CONSOLIDATE statement is executed automatically at the remote database.

**Adaptive Server Anywhere example**

The following Adaptive Server Anywhere statement grants CONSOLIDATE permissions to the `hq_user` user ID, using the VIM e-mail system:

```sql
GRANT CONSOLIDATE TO hq_user
TYPE vim
ADDRESS 'hq_address'
```

There is no SEND clause in this statement, so the default is used and messages will be sent to the consolidated database every time the Message Agent is run.
Adaptive Server Enterprise example

The following Adaptive Server Enterprise statement grants CONSOLIDATE permissions to user hq_user:

```
exec sp_grant_consolidate 'hq_user', address
go
```

Revoking REMOTE and CONSOLIDATE permissions

A user can be removed from a SQL Remote installation by revoking their REMOTE permissions. Revoking remote user permissions also drops any subscriptions for that user.

Revoking permissions from Sybase Central

You can revoke REMOTE permissions on both Adaptive Server Enterprise and Adaptive Server Anywhere from Sybase Central.

To revoke REMOTE permissions from Sybase Central:

1. Open the SQL Remote folder.
2. Open the Remote Users folder.
3. Right-click the user whose permission you wish to revoke, and select Revoke Remote from the popup menu.

Revoking permissions in Adaptive Server Anywhere

REMOTE and CONSOLIDATE permissions can be revoked from a user using the REVOKE statement. The following statement revokes REMOTE permission from user S_Beaulieu.

```
REVOKE REMOTE FROM S_Beaulieu
```

DBA authority is required to revoke REMOTE or CONSOLIDATE access.

Revoking permissions in Adaptive Server Enterprise

REMOTE permissions can be revoked from a user using the sp_revoke_remote procedure. This procedure takes a single argument, which is the user ID of the user. The following statement revokes REMOTE permission from user S_Beaulieu.

```
exec sp_revoke_remote 'S_Beaulieu'
```

```
go
```
Assigning permissions in multi-tier installations

Special considerations are needed for assigning permissions in multi-tier installations. The permissions in a three-level SQL Remote setup are summarized in the following diagrams. In each diagram one database is shaded; the diagram shows the permissions that need to be granted in that database for the user ID representing each of the other databases. The phrase "No permissions" means that the database is not granted any permissions in the shaded database.

The following picture shows SQL Remote permissions, as granted at the consolidated site of a three-tier installation.

The following picture shows SQL Remote permissions, as granted at an internal site of a three-tier installation.

The following picture shows SQL Remote permissions, as granted at an internal site of a three-tier installation.
Granting the appropriate PUBLISH and CONSOLIDATE permissions at remote databases is done automatically by the database extraction utility.
Using message types

SQL Remote supports several different systems for exchanging messages. The message systems supported by SQL Remote are:

- **file**  Storage of message files in directories on a shared file system for reading by other databases.
- **ftp**  Storage of message files in directories accessible by a file transfer protocol (ftp) link.
- **mapi**  Microsoft's messaging API (MAPI) link, used in Microsoft Mail and other electronic mail systems.
- **smtp**  Internet Simple Mail Transfer Protocol (SMTP/POP), used in Internet e-mail.
- **vim**  Lotus's Vendor Independent Messaging (VIM), used in Lotus Notes and cc:Mail.

A database can exchange messages using one or more of the available message systems.

Operating system availability

Not all message systems are supported on all operating systems for which SQL Remote is available. The links are implemented as DLLs on Windows 3.x, Windows 95, and Windows NT.

The Message Agent for UNIX and NetWare operating systems supports the **file** link only, as a compiled library.

For more information

For full details on operating system availability, see the section on each message system:

- For more information on the **file** message system, see "The file sharing message system" on page 232.
- For more information on the **ftp** message system, see "The ftp message system" on page 233.
- For more information on the **smtp** message system, see "The SMTP message system" on page 233.
- For more information on the **mapi** message system, see "The MAPI message system" on page 235.
- For more information on the **vim** message system, see "The VIM message system" on page 237.
Working with message types

Each message type definition includes the type name (file, ftp, smtp, mapi, or vim) and also the address of the publisher under that message type. The publisher address at a consolidated database is used by the database extraction utility as a return address when creating remote databases. It is also used by the Message Agent to identify where to look for incoming messages for the file system.

The address supplied with a message type definition is closely tied to the publisher ID of the database. Valid addresses are considered in following sections.

Before you can use a message system, you must set the publisher's address.

Using Sybase Central to work with message types

You can create and alter message types in Sybase Central.

- **To add a message type, using Sybase Central:**
  1. In the left pane, open the Message Types folder. The Message Types folder is inside the SQL Remote folder.
  2. Double-click Add Message Type. The New Message Type window appears.
  3. Enter one of the existing message type names, and a publisher address, in the appropriate fields. Click OK to save the definition in the database.

     If you wish to change the publisher's address, you can do so by altering a message type.

- **To alter a message type, using Sybase Central:**
  1. In the left pane, open the Message Types folder. The Message Types folder is inside the SQL Remote folder.
  2. In the right pane, right-click the message type you wish to alter and select Properties from the popup menu. The Message Type Properties window appears.
  3. Enter a new publisher address, in the appropriate fields. Click OK to save the definition in the database.

     If you wish to drop a message type from the installation, you can do so.
Using message types

To drop a message type, using Sybase Central:

1. In the left pane, open the Message Type folder. The Message Type folder is inside the SQL Remote folder.
2. In the right pane, right-click the message type you wish to alter and select Delete from the popup menu.

Using commands to work with message types

To create a message type:

1. Make sure you have decided on an address for the publisher under the message type.
2. Enter the command to create the message type.

For Adaptive Server Anywhere, use the CREATE REMOTE MESSAGE TYPE statement. This statement has the following syntax:

```
CREATE REMOTE MESSAGE TYPE type-name
ADDRESS address-string
```

For Adaptive Server Enterprise, use the `sp_remote_type` procedure. This procedure takes the following arguments:

```
sp_remote_type type-name, address-string
```

In these statements, `type-name` is one of the message systems supported by SQL Remote, and `address-string` is the publisher's address under that message system.

If you wish to change the publisher's address, you can do so by altering the message type.

To alter a message type:

1. Make sure you have decided on a new address for the publisher under the message type.
2. Enter the command to alter the message type.

For Adaptive Server Anywhere, use the ALTER REMOTE MESSAGE TYPE statement. This statement has the following syntax:

```
ALTER REMOTE MESSAGE TYPE type-name
ADDRESS address-string
```

For Adaptive Server Enterprise, use the `sp_remote_type` procedure in the same way as creating a message type. This procedure takes the following arguments:

```
sp_remote_type type-name, address-string
```
In these statements, \textit{type-name} is one of the message systems supported by SQL Remote, and \textit{address-string} is the publisher's address under that message system.

You can also drop message types if they are no longer used in your installation. This has the effect of removing the publisher's address from the definition.

\begin{itemize}
\item \textbf{To drop a message type:}
\end{itemize}

\begin{itemize}
\item Enter the command to drop the message type.
\end{itemize}

For Adaptive Server Anywhere, use the DROP REMOTE MESSAGE TYPE statement. This statement has the following syntax:

\begin{verbatim}
DROP REMOTE MESSAGE TYPE type-name
\end{verbatim}

For Adaptive Server Enterprise, use the \texttt{sp\_drop\_remote\_type} procedure in the same way as creating a message type. This procedure takes the following arguments:

\begin{verbatim}
sp_drop_remote_type type-name
\end{verbatim}

In these statements, \textit{type-name} is one of the message systems supported by SQL Remote.

\section*{Setting message type control parameters}

Each message link has several parameters that govern aspects of its behavior. The following sections document these parameters.

The message link control parameters are stored in the following places:

\begin{itemize}
\item \textbf{Windows 95 and Windows NT} In the registry, at the following location:

\begin{verbatim}
\\%HKEY_CURRENT_USER\%Software\%Sybase\%SQL Remote
\end{verbatim}

\item \textbf{Windows 3.x} In the file SQLANY.INI, in your SQL Remote installation directory.

\item \textbf{NetWare} You should create a file named \texttt{dbremote.ini} in the \texttt{sys:system} directory to hold the FILE system directory setting.

\item \textbf{UNIX} The FILE system directory setting is held in the SQLREMOTE environment variable.
\end{itemize}

The \texttt{sqlremote} environment variable holds a path that can be used as an alternative to one of the control parameters for the file sharing system.
The parameters available for each message system are discussed in the following sections. Each section describes a single message system.

### The file sharing message system

SQL Remote can be used even if you do not have a message system in place, by using the **file** message system.

#### Supported operating systems

The file sharing message system is supported on all platforms for which SQL Remote is available, for both Adaptive Server Enterprise and Adaptive Server Anywhere.

#### Addresses in the file message system

The **file** message system is a simple file-sharing system. A **file** address for a remote user is a subdirectory into which all their messages are written. To retrieve messages from their "inbox", an application reads the messages from the directory containing the user's files. Return messages are sent to the address (written to the directory) of the consolidated database.

You can also use the **file** system to put the messages in directories on the consolidated and remote machines. A simple file transfer mechanism can then be used to exchange the files periodically to effect replication.

#### Root directory for addresses

The **file** system addresses are typically subdirectories of a shared directory that is available to all SQL Remote users, whether by modem or on a local area network. Each user should have a registry entry, initialization file entry, or SQLREMOTE environment variable pointing to the shared directory.

### FILE message control parameters

The FILE message system uses the following control parameters:

- **Directory** This is set to the directory under which the messages are stored. The setting is an alternative to the SQLREMOTE environment variable.

- **Debug** This is set to either YES or NO, with the default being NO. When set to YES, all file system calls made by the FILE link are displayed.

The FILE section of the sqlany.ini file (Windows 3.x) has the following entries:

```
[ FILE ]
Directory=path
Debug={ yes | no }
```

On NetWare, you should create a file named dbremote.ini in the sys:\system directory to hold the directory setting.
The ftp message system

Supported operating systems
The ftp message system is supported on the Windows NT, Windows 95, and Windows 3.x operating systems.

Addresses for ftp
In the ftp message system, messages are stored in directories under a root directory on an ftp host. The ftp host and the root directory are specified by message system control parameters held in the registry or initialization file, and the address of each user is the subdirectory where their messages are held.

FTP message control parameters

The ftp message system uses the following control parameters:

♦ host The host name of the computer where the messages are stored. This can be a host name (such as ftp.powersoft.com) or an IP address (such as 192.138.151.66).

♦ user The user name for accessing the ftp host.

♦ password The password for accessing the ftp host.

♦ root_directory The root directory within the ftp host site, under which the messages are stored.

♦ port Usually not required. This is the IP port number used for the Ftp connection.

♦ debug This is set to either YES or NO, with the default being NO. When set to YES, debugging output is displayed.

The SMTP message system

The Simple Mail Transfer Protocol (SMTP) is used in Internet e-mail products.

With the SMTP system, SQL Remote sends messages using Internet mail. The messages are encoded to a text format and sent in an e-mail message to the target database. The messages are sent using an SMTP server, and retrieved from a POP server: this is the way that many e-mail programs send and receive messages.

Supported operating systems
The SMTP message system is supported on the following operating systems:

♦ Windows 95
♦ Windows NT
♦ Windows 3.x
Using message types

SMTP addresses and user IDs

To use SQL Remote and an SMTP message system, each database participating in the setup requires a SMTP address, and a POP3 user ID and password. These are distinct identifiers: the SMTP address is the destination of each message, and the POP3 user ID and password are the name and password entered by a user when they connect to their mail box.

**Separate e-mail account recommended**

It is recommended that a separate POP e-mail account be used for SQL Remote messages.

Sharing SMTP/POP addresses

The database should have its own e-mail account for SQL Remote messages, separate from personal e-mail messages intended for reading. This is because many e-mail readers will collect e-mail in the following manner:

1. Connect to the POP Host and download all messages.
2. Delete all messages from POP Host
3. Disconnect from POP Host.
4. Read mail from the local file or from memory

This causes a problem, as the e-mail program downloads and deletes all of the SQL Remote e-mail messages as well as personal messages. If you are certain that your e-mail program will not delete unread messages from the POP Host then you may share an e-mail address with the database as long as you take care not to delete or alter the database messages.

These messages are easy to recognize, as they are filled with lines of seemingly random text.

SMTP message control parameters

Before the Message Agent connects to the message system to send or receive messages, the user must either have a set of control parameters already set on their machine, or must fill in a window with the needed information. This information is needed only on the first connection. It is saved and used as the default entries on subsequent connects.

The SMTP message system uses the following control parameters:
The MAPI message system

The Message Application Programming Interface (MAPI) is used in several popular e-mail systems, such as Microsoft Mail and later versions of Lotus cc:Mail. SQL Remote supports the MAPI message system under Windows 3.x, Windows 95, and Windows NT.
### Using message types

#### Supported operating systems

The MAPI message system is supported on the following operating systems:

- Windows 95
- Windows NT
- Windows 3.x

#### MAPI addresses and user IDs

To use SQL Remote and a MAPI message system, each database participating in the setup requires a MAPI user ID and address. These are distinct identifiers: the MAPI address is the destination of each message, and the MAPI user ID is the name entered by a user when they connect to their mail box.

#### MAPI message and the e-mail inbox

Although SQL Remote messages may arrive in the same mail box as e-mail intended for reading, they do not in general show up in your e-mail inbox.

SQL Remote sends application-defined messages, which MAPI identifies and hides when the mailbox is opened. In this way, users can use the same e-mail address and same connection to receive their personal e-mail and their database updates, yet the SQL Remote messages do not interfere with the mail intended for reading.

If a message is routed via the Internet, the special message type information is lost. The message then does show up in the recipient's mailbox.

#### MAPI message control parameters

The MAPI message system uses the following control parameters:

- **Debug** When set to YES, displays all MAPI calls and the return codes. This is useful for troubleshooting MAPI support problems. Default is NO.

- **Force Download** (default YES) controls if the MAPI_FORCE_DOWNLOAD flag is set when calling MapiLogon. This might be useful when using remote mail software that dials when this flag is set.

- **IPM_Receive** This can be set to YES or NO (default NO). If set to YES, the MAPI link receives IPM messages, which are visible in the mailbox. If set to NO, the MAPI link receives IPC messages, which are not visible in the mailbox. This may be useful if your MAPI provider does not support IPC messages. Also, it may be useful when receiving messages over the Internet. In this case, the sender might not be using MAPI or the IPC attributes are have been lost.


- **IPM_Send**  This can be set to YES or NO (default NO). If set to YES, the MAPI link sends IPM messages, which are visible in the mailbox. If set to NO, the MAPI link sends IPC messages, which are not visible in the mailbox. This may be useful if your MAPI provider does not support IPC messages.

- **Profile**  Use the specified Microsoft Exchange profile. You should use this if you are running the Message Agent as a service.

The MAPI section of the *sqlany.ini* file (Windows 3.x) has the following entries:

```plaintext
[ MAPI ]
IPM_Send={yes | no}
IPM_Receive={ yes | no }
Force_Download={yes | no }
Debug={yes | no}
```

### The VIM message system

The Vendor Independent Messaging system (VIM) is used in Lotus Notes and in some releases of Lotus cc:Mail.

The VIM message system is supported on the following operating systems:

- Windows 95
- Windows NT
- Windows 3.x

To use SQL Remote and a VIM message system, each database participating in the setup requires a VIM user ID and address. These are distinct identifiers: the VIM address is the destination of each message, and the VIM user ID is the name entered by a user when they connect to their mail box.

### VIM message control parameters

The VIM message system uses the following control parameters:

- **Path**  This corresponds to the Path field in the cc:Mail login window. It is not applicable to and is ignored under Lotus Notes.

- **Userid**  This corresponds to the User ID field in the cc:Mail login window.

- **Password**  This corresponds to the Password field in the cc:Mail login window. If all of Path, Userid, and Password are set, the login window is not displayed.
**Using message types**

- **Debug**  When set to YES, displays all VIM calls and the return codes. This is useful for troubleshooting VIM support problems. Default is NO.

- **Receive_All**  When set to YES, the Message Agent checks all messages to see if they are SQL Remote messages. When set to NO (the default), the Message Agent looks only for messages of the application-defined type `SQLRemoteData`. This leads to improved performance in Notes.

  Setting `ReceiveAll` to YES is useful in setups where the message type is lost, reset, or never set. This includes setups including cc:Mail messages, or over the Internet.

- **Send_VIM_Mail**  When set to YES, the Message Agent sends messages compatible with Adaptive Server Anywhere releases before 5.5.01, and compatible with cc:Mail. If this is set to YES, you should ensure that `Receive_All` is set to YES also.

The VIM section of the `sqlany.ini` file (Windows 3.x) has the following entries:

```
[ VIM ]
Path=path
Userid=userid
Password=password
Debug={yes | no}
Receive_All = {yes | no}
Send_VIM_Mail = {yes | no}
```
Running the Message Agent

The SQL Remote Message Agent is a key component in SQL Remote replication. The Message Agent handles both the sending and receiving of messages. It carries out the following functions:

- It processes incoming messages, and applies them in the proper order to the database.
- It scans the transaction log or stable queue at each publisher database, and translates the log entries into messages for subscribers.
- It parcels the log entries up into messages no larger than a fixed maximum size (50,000 bytes by default), and sends them to subscribers.
- It maintains the message tracking information in the system tables, and manages the guaranteed transmission mechanism.

Executable names

The Message Agent for Adaptive Server Enterprise is named ssremote.exe, and the Message Agent for Adaptive Server Anywhere is named dbremote.exe.

The Message Agent for Adaptive Server Enterprise uses a stable queue to hold transactions until they are no longer needed. For more information on the stable queue, see "How the Message Agent for Adaptive Server Enterprise works" on page 278.

Message Agent batch and continuous modes

The Message Agent can be run in one of two modes:

- **Batch mode** In batch mode, the Message Agent starts, receives and sends all messages that can be received and sent, and then shuts down.

  Batch mode is useful at occasionally-connected remote sites, where messages can only be exchanged with the consolidated database when the connection is made: for example, when the remote site dials up to the main network.

- **Continuous mode** In continuous mode, the Message Agent periodically sends messages, at times specified in the properties of each remote user. When it is not sending messages, it receives messages as they arrive.

  Continuous mode is useful at consolidated sites, where messages may be coming in and going out at any time, to spread out the workload and to ensure prompt replication.
The options available depend on the send frequency options selected for the remote users. Sending frequency options are described in "Selecting a send frequency" on page 223.

❖ To run the Message Agent in continuous mode:

1. Ensure that every user has a sending frequency specified. The sending frequency is specified by a SEND AT or SEND EVERY option in the GRANT REMOTE statement (Adaptive Server Anywhere) or sp_grant_remote procedure (Adaptive Server Enterprise).
2. Start the Message Agent without using the −b command-line switch.

❖ To run the Message Agent in batch mode:

♦ Either:
  ♦ Have at least one remote user who has neither a SEND AT nor a SEND EVERY option in their remote properties, or
  ♦ Start the Message Agent using the −b command-line switch.

Connections used by the Message Agent

The Message Agent uses a number of connections to the database server. These are:

♦ One global connection, alive all the time the Message Agent is running.
♦ One connection for scanning the log. This connection is alive during the scan phase only.
♦ One connection for executing commands from the log-scanning thread. This connection is alive during the scan phase only.
♦ One connection for the stable queue (Adaptive Server Enterprise only). This connection is alive during the scan and send phases.
♦ One connection for processing synchronize subscription requests. This connection is alive during the send phase only.
♦ One connection for each worker thread. These connections are alive during the receive phase only.
Replication system recovery procedures

SQL Remote replication places new requirements on data recovery practices at consolidated database sites. Standard backup and recovery procedures enable recovery of data from system or media failure. In a replication installation, even if such recovery is achieved, the recovered database can be out of synch with remote databases. This can require a complete resynchronization of remote databases, which can be a formidable task if the installation involves large numbers of databases.

In short, recovery of the consolidated database from a failure at the consolidated site is only part of the task of recovering the entire replication installation.

Protection of the replication system against media failures has two aspects:

- **Backup and log management** Solid backup procedures and log management procedures for the consolidated database server are an essential part of recovery plans. Backup procedures protect against media failure on the database device. Using a transaction log mirror protects against media failure on the transaction log device.

  For more information about backup and log management procedures, see the sections "Transaction log and backup management" on page 262 and "Adaptive Server Enterprise transaction log and backup management" on page 286.

- **Message Agent configuration** The Message Agent command-line options provide ways for you to tune Message Agent behavior to match your backup and recovery requirements.

  Message Agent configuration is discussed in the following pages.

By default, the Message Agent processes all committed transactions. When the Message Agent is run with the –u command-line switch, only transactions that have been backed up by the database backup commands are processed.

For Adaptive Server Anywhere, transaction log backup is carried out using Sybase Central or the dbbackup command-line utility, or off-line copying and renaming of the log file. For Adaptive Server Enterprise, transaction log backup is carried out using the dump transaction statement.

By sending only backed-up transactions, the replication installation is protected against media failure on the transaction log. Maintaining a mirrored transaction log also accomplishes this goal.

The –u switch provides additional protection against total site failure, if backups are carried out to another site.
Ensuring consistent Message Agent settings

Some Message Agent settings need to be the same throughout an installation, and so should be set before deployment. This section lists the settings that need to be the same.

♦ Maximum message length  The maximum message length for SQL Remote messages has a default value of 50K. This is configurable, using the Message Agent -1 command-line switch. However, the maximum message length must be the same for each Message Agent in the installation, and may be restricted by operating system memory allocation limits.

  Received messages that are longer than the limit are deleted as corrupt messages.

☞ For details of this setting, see "The Message Agent" on page 306.

The Message Agent and replication security

Messages sent by the SQL Remote Message Agent have a very simple encryption that protects against casual snooping. However, the encryption scheme is not intended to provide full protection against determined efforts to decipher them.
Tuning Message Agent performance

Who needs to read this section? If performance is not a problem at your site, you do not need to read this section.

There are several options you can use to tune the performance of the Message Agent. This section describes those options.

Sending messages and receiving messages are two separate processes. The major performance issues for these two processes are different.

- **Replication throughput**  The major bottleneck for total throughput of SQL Remote sites is generally receiving messages from many remote databases and applying them to the database at the consolidated site. You can control this step by tuning the receive process of the Message Agent at the consolidated site.

- **Replication turnaround**  The time lag from when data is entered at one site to when it appears at other sites is the turnaround time for replication. You can control this time lag.

Tuning throughput by controlling Message Agent threading

It is assumed in this section that you are tuning the performance of a Message Agent that is running in continuous mode at a consolidated site.

Worker threads can be used by the Message Agent to apply incoming messages from remote users. This can improve throughput by allowing messages to be applied in parallel rather than serially.

<table>
<thead>
<tr>
<th>Setting the number of worker threads</th>
<th>Performance benefits from worker threads</th>
</tr>
</thead>
</table>
| The number of worker threads is set on the Message Agent command line, using the `-w` switch. The following command line starts the Message Agent for Adaptive Server Enterprise with twenty worker threads applying messages:

```
ssremote -c "eng=..." -w 20
```

The default is to use no worker threads, so that all messages are applied serially. The maximum number of worker threads is 50.

For the Message Agent for Adaptive Server Anywhere, the performance advantage will be most significant when the server is on a system with a striped drive array.

For Adaptive Server Enterprise, the Message Agent will benefit even more if the Server is used with multiple engines configured.
### Tuning Message Agent performance

**What messages are applied in parallel**

When worker threads are being used, messages from different remote users are applied in parallel. Messages from a single remote user are applied serially. For example, ten messages from a single remote user will be applied by a single worker thread in the correct order.

Deadlock is handled by re-applying the rolled back transaction at a later time.

Reading messages from the message system is single-threaded. Messages are read and the header information is examined (to determine the remote user and the correct order of application) before passing them off to worker threads to be applied.

Building messages and sending messages is single-threaded.

**Open Client version**

To use multiple worker threads with the Adaptive Server Enterprise Message Agent, you need to be using Open Client version 11.1 or above.

The Message Agent prints a message and then does not use worker threads when pre-11.1 versions are being used. The Open Client version is displayed in the first few lines of the Message Agent output.

### Tuning throughput by caching messages

The Message Agent caches incoming messages in a configurable area of memory as it reads them.

**Specifying the message cache size**

The size of the message cache is specified on the Message Agent command line, using the `-m` command-line switch.

The `-m` option specifies the maximum amount of memory to be used by the Message Agent for building messages. The allowed size can be specified as `n` (in bytes), `nK`, or `nM`. The default is 2048K (2M).

**Example**

The following command line starts an Adaptive Server Anywhere Message Agent using twelve Megabytes of memory as a message cache:

```
dbremote -c "eng=..." -m 12M
```

**How messages are cached**

When transactions are large, or messages arrive out of order, they are stored in memory by the Message Agent until the message is to be applied. This caching of messages prevents rereading of out-of-order messages from the message system, which may lower performance on large installations. It is especially important when messages are being read over a WAN (such as Remote Access Services or POP3 through a modem). It also avoids contention between worker threads reading messages (a single threaded task) because the message contents are cached.
When the memory usage specified using the \texttt{-m} switch is exceeded, messages are flushed in a least-recently-used fashion.

This switch is provided primarily for customers considering a single consolidated database for thousands of remote databases.

\textbf{Tuning incoming message polling}

When running a Message Agent in continuous mode, typically at a consolidated database site, you can control how often it polls for incoming messages, and how "patient" it is in waiting for messages that arrive out of order before requesting that the message be resent. Tuning these aspects of the behavior can have a significant effect on performance in some circumstances.

\textbf{Issues to consider}

The issues to consider when tuning the message-receiving process are similar to those when tuning the message-sending process.

\begin{itemize}
\item \textbf{Regular messages} \hspace{1cm} Your choices dictate how often the Message Agent polls for incoming messages from remote databases.
\item \textbf{Resend requests} \hspace{1cm} You can control how many polls to wait until an out-of-order message arrives, before requesting that it be resent.
\item \textbf{Processing incoming messages} \hspace{1cm} If your polling period for incoming messages is too long, compared to the frequency with which messages are arriving, you could end up with messages sitting in the queue, waiting to be processed. If your polling period is too short, you will waste resources polling when no messages are in the queue.
\end{itemize}

\(\text{\copyright} \) For more information on the message sending process, see "Tuning the message sending process" on page 247.

\textbf{Polling interval}

By default, a Message Agent running in continuous mode polls one minute after finishing the previous poll, to see whether new messages have arrived. You can configure the polling interval using the \texttt{-rd} command-line option.

The default polling interval from the end of one poll to the start of another is one minute. You can poll more frequently using a value in seconds, as in the following command line:

\begin{verbatim}
dbremote -rd 30s
\end{verbatim}

Alternatively, you can poll less frequently, as in the following command line, which polls every five minutes:

\begin{verbatim}
dbremote -rd 5
\end{verbatim}
Tuning Message Agent performance

Setting a very small interval may have some detrimental impact on overall system throughput, for the following reasons:

♦ Each poll of the mail server (if you are using e-mail) places a load on your message system. Too-frequent polling may affect your message system and produce no benefits.

♦ If you do not modify the Message Agent patience before it assumes that an out of sequence message is lost, and requests it be sent again, you can flood your system with resend requests.

In general, you should not use a very small polling interval unless you have a specific reason for requiring a very quick response time for messages.

Setting larger intervals may provide a better overall throughput of messages in your system, at the cost of waiting somewhat longer for each message to be applied. In many SQL Remote installations, optimizing turnaround time is not the primary concern.

Requesting resends

If, when the Message Agent polls for incoming messages, one message is missing from a sequence, the Message Agent does not immediately request that the message be resent. Instead, it has a default patience of one poll.

If the next message expected is number 6 and message 7 is found, the Message Agent takes no action until the next poll. Then, if no new message for that user is found, it issues a resend request.

You can change the number of polls for which the Message Agent waits before sending a request using the -rp command-line option. This option is often used in conjunction with the -rd option that sets the polling interval.

For example, if you have a very small polling interval, and a message system that does not preserver the order in which messages arrive, it may be very common for out-of-sync messages to arrive only after two or three polls have been completed. In such a case, you should instruct the Message Agent to be more patient before sending a resend request, by increasing the -rp value. If you do not do this, a large number of unnecessary resend requests may be sent.

Example

Suppose there are two remote users, named user1 and user2, and suppose the Message Agent command line is as follows:

dbremote -rd 30s -rp 3
In the following sequence of operations, messages are marked as user\(X.n\) so that user1.5 is the fifth message from user1. The Message Agent expects messages to start at number 1 for both users.

At time 0 seconds:
1 The Message Agent reads user1.1, user2.4
2 The Message Agent applies user1.1
3 The Message Agent patience is now user1: N/A, user2: 3, as an out of sequence message has arrived from user 2.

At time 30 seconds:
1 The Message Agent reads: no new messages
2 The Message Agent applies: none
3 The Message Agent patience is now user1: N/A, user2: 2

At time 60 seconds:
1 The Message Agent reads: user1.3
2 The Message Agent applies: no new messages
3 The Message Agent patience: user1: 3, user2: 1

At time 90 seconds:
1 The Message Agent reads: user1.4
2 The Message Agent applies: none
3 The Message Agent patience user1: 3, user2: 0
4 The Message Agent issues resend to user2.

When a user receives a new message, it resets the Message Agent patience even if that message is not the one expected.

### Tuning the message sending process

The turnaround time for replication is governed by how often each sites sends messages and how often each site polls for incoming messages. To achieve a small time lag between data entry and data replication, you can set a small value for the \(--\text{fwd}\) Message Agent command-line option, which controls the frequency for polling to see if more data needs to be sent.

#### Issues to consider

The issues to consider when tuning the message-sending process are similar to those when tuning the incoming-message polling frequency:
Tuning Message Agent performance

- **Regular messages** Your choices dictate how often updates are sent to remote databases.
- **Resend requests** When a remote user requests that a message be resent, the Message Agent needs to take special action that can interrupt regular message sending. You can control the urgency with which these resend requests are processed.
- **Number and size of messages** If you send messages very frequently, there is more chance of small messages being sent. Sending messages less frequently allows more instructions to be grouped in a single message. If a large number of small messages is a concern for your message system, then you may have to avoid using very small polling periods.

🔗 For more information on tuning polling for the incoming-messages, see "Tuning incoming message polling" on page 245.

Polling interval

You control the interval to wait between polls for more data from the transaction log to send using the `-sd` command-line option, which has a default of one minute. The following example sets the polling interval to 30 seconds:

```
  dbremote -sd 30s ...
```

Alternatively, you can poll less frequently, as in the following command line, which polls every five minutes:

```
  dbremote -sd 5
```

Setting a very small interval may have some detrimental impact on overall system throughput, for the following reasons:

- Too-frequent polling produces many short messages. If the message load places a strain on your message system, throughput could be affected.

Setting larger intervals may provide a better overall throughput of messages in your system, at the cost of waiting somewhat longer for each message to be applied. In many SQL Remote installations, optimizing turnaround time is not the primary concern.
Resending messages

When a user requests that a message be resent, the message has to be retrieved from early in the transaction log. Going back in the transaction log to retrieve this message and send it causes the Message Agent to interrupt the regular sending process. If you are tuning your SQL Remote installation for optimum performance, you must balance the urgency of sending requests for resent messages with the priority of processing regular messages.

The `-ru` command-line option controls the urgency of the resend requests. The value for the parameter is a time in minutes (or in other units if you add `s` or `h` to the end of the number), with a default of zero.

To help the Message Agent delay processing resend requests until more have arrived before interrupting the regular message sending activity, set this option to a longer time.

The following command line waits one hour until processing a resend request.

```
dbremote -ru 1h ...
```
Encoding and compressing messages

As messages pass through e-mail and other message systems, there is a danger of them becoming corrupted. For example, some message systems use certain characters or character combinations as control characters.

Message size affects the efficiency with which messages pass through a system. Compressed messages can be processed more efficiently by a message system than uncompressed messages. On the other hand, carrying out compression can itself take a significant amount of time.

SQL Remote has a message encoding and compression scheme built in to the Message Agent. The scheme provides the following features:

- **Compatibility** The system can be set up to be compatible with previous versions of the software.
- **Compression** You can select a level of compression for your messages.
- **Encoding** SQL Remote encodes messages to ensure that they pass through message systems uncorrupted. The encoding scheme can be customized to provide extra features.

To be compatible with previous versions of the software, you should set the database option COMPRESSION to be -1 (minus one) at each database running the Version 6 software. This setting ensures that messages are sent out in a format compatible with older versions of the software.

If you upgrade the consolidated database Message Agent first, you should set its COMPRESSION database option to -1. As each remote site in your replication system is upgraded to Version 6, you can change its setting of the COMPRESSION option to a value between 0 (no compression) and 9 (maximum compression). This allows you to take advantage of compression features on messages being sent to the consolidated database. Once all remote sites are upgraded, you can set the consolidated site Message Agent COMPRESSION option to a value other than -1.

In addition, setting COMPRESSION to a value other than -1 allows you to take advantage of the Version 6 message encoding improvements.

The encoding scheme

The default message-encoding behavior of SQL Remote is as follows:

- For message systems that can use binary message formats, no encoding is carried out.
Some message systems, including SMTP, VIM, and MAPI, require text-based message formats. For these systems, an encoding DLL (dbencod.dll and ssencod.dll) translates messages into a text format before sending. The message format is unencoded at the receiving end using the same DLL.

You can instruct SQL Remote to use a custom encoding scheme. The tools for building a custom encoding scheme are described in the following section.

If the COMPRESSometown database option is set to -1, then a Version 5 compatible encoding is carried out for all message systems.

Creating custom encoding schemes

You can implement a custom encoding scheme by building a custom encoding DLL. You could use this DLL to apply special features required for a particular messages system, or to collect statistics, such as how many messages or how many bytes were sent to each user.

The header file dbhmt.h, installed into the h subdirectory of your installation directory, provides an application programming interface for building such a scheme.

To instruct SQL Remote to use your DLL for a particular message system, you must make a registry entry for that system. The registry entry should be made in the following location:

```
Software
 \Sybase
  \SQL Remote
   \message-system
    \encode_dll
```

where message-system is one of the SQL Remote message systems (file, smtp, and so on). You should set this registry entry to the name of your encoding DLL.

**Encoding and decoding must be compatible**

If you implement a custom encoding, you must make sure that the DLL is present at the receiving end, and that the DLL is in place to decode your messages properly.
The message tracking system

SQL Remote has a message tracking system to ensure that all replicated operations are applied in the correct order, no operations are missed, and no operation is applied twice.

Message system failures may lead to replication messages not reaching their destination, or reaching it in a corrupt state. Also, messages may arrive at their destination in a different order from that in which they were sent. This section describes the SQL Remote system for detecting and correcting message system errors, and for ensuring correct application of messages.

If you are using an e-mail message system, you should confirm that e-mail is working properly between the two machines if SQL Remote messages are not being sent and received properly.

The SQL Remote message tracking system is based on status information maintained in the remoteuser SQL Remote system table. The table is maintained by the Message Agent. The Message Agent at a subscriber database sends confirmation to the publisher database to ensure that remoteuser is maintained properly at each end of the subscription.

For Adaptive Server Anywhere, the remoteuser table is the sys.sysremoteuser system table. For Adaptive Server Enterprise, this is the sr_remoteuser table.

Status information in the remoteuser table

The remoteuser SQL Remote system table contains a row for each subscriber, with status information for messages sent to and received by that subscriber. At the consolidated database, remoteuser contains a row for each remote user. At each remote database, remoteuser contains a single row maintaining information for the consolidated database. (Recall that the consolidated database subscribes to publications from the remote database.)

The remoteuser SQL Remote system table at each end of a subscription is maintained by the Message Agent.

Tracking messages by transaction log offsets

The message-tracking status information takes the form of offsets in the transaction logs of the publisher and subscriber databases. Each COMMIT is marked in the transaction log by a well-defined offset. The order of transactions can be determined by comparing their offset values.
Message ordering  
When messages are sent, they are ordered by the offset of the last COMMIT of the preceding message. If a transaction spans several messages, there is a serial number within the transaction to order the messages correctly. The default maximum message size is 50,000 bytes, but you can use the Message Agent -1 command-line switch to change this setting.

Sending messages  
The log_sent column holds the local transaction log offset for the latest message sent to the subscriber. When the Message Agent sends a message, it sets the log_sent value to the offset of the last COMMIT in the message. Once the message has been received and applied at the subscribed database, confirmation is sent back to the publisher. When the publisher Message Agent receives the confirmation, it sets the confirm_sent column for that subscriber with the local transaction log offset. Both log_sent and confirm_sent are offsets in the local database transaction log, and confirm_sent cannot be a later offset than log_sent.

Receiving messages  
When the Message Agent at a subscriber database receives and applies a replication update, it updates the log_received column with the offset of the last COMMIT in the message. The log_received column at any subscriber database therefore contains a transaction log offset in the publisher database's transaction log. After the operations have been received and applied, the Message Agent sends confirmation back to the publisher database and also sets the confirm_received value in the local SYSPERUSER table. The confirm_received column at any subscriber database contains a transaction log offset in the publisher database's transaction log.

Subscriptions are two-way  
SQL Remote subscriptions are two-way operations: each remote database is a subscriber to publications of the consolidated database and the consolidated database subscribes to a matching publication from each remote database. Therefore, the remoteduser SQL Remote system tables at the consolidated and remote database hold complementary information.

The Message Agent applies transactions and updates the log_received value atomically. If a message contains several transactions, and a failure occurs while a message is being applied, the log_received value corresponds exactly to what has been applied and committed.

Resending messages  
The remoteduser SQL Remote table contains two other columns that handle resending messages. The resend_count and rereceive_count columns are retry counts that are incremented when messages get lost or deleted for some reason.

The log_send column is updated by the Message Agent based on the SEND frequency information (SEND AT or SEND EVERY as specified in the GRANT REMOTE statement or sp_grant_remote procedure). The value of log_send must be greater than that of log_sent for a user in order for messages to be sent to that user.
Handling of lost or corrupt messages

When messages are received at a subscriber database, the Message Agent applies them in the correct order (determined from the log offsets) and sends confirmation to the publisher. If a message is missing, the Message Agent increments the local value of \textit{rereceive\_count}, and requests that it be resent. Other messages present or en route are not applied.

The request from a subscriber to resend a message increments the \textit{resend\_count} value at the publisher database, and also sets the publisher's \textit{log\_sent} value to the value of \textit{confirm\_sent}. This resetting of the \textit{log\_sent} value causes operations to be resent.

<table>
<thead>
<tr>
<th><strong>Users cannot reset log_sent</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>The \textit{log_sent} value cannot be reset by a user, as it is in a system table.</td>
</tr>
</tbody>
</table>

Each message is identified by three values:
- Its \textit{resend\_count}.
- The transaction log offset of the last COMMIT in the previous message.
- A serial number within transactions, for transactions that span messages.

Messages with a \textit{resend\_count} value smaller than \textit{rereceive\_count} are not applied; they are deleted. This ensures that operations are not applied more than once.
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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running the Message Agent</td>
<td>256</td>
</tr>
<tr>
<td>Error reporting and handling</td>
<td>258</td>
</tr>
<tr>
<td>Transaction log and backup management</td>
<td>262</td>
</tr>
<tr>
<td>Using passthrough mode</td>
<td>273</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Verbose keyword</th>
<th>Short form</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatabaseFile</td>
<td>DBF</td>
<td>string</td>
</tr>
<tr>
<td>DatabaseName</td>
<td>DBN</td>
<td>string</td>
</tr>
<tr>
<td>DatabaseSwitches</td>
<td>DBS</td>
<td>string</td>
</tr>
<tr>
<td>EngineName</td>
<td>ENG</td>
<td>string</td>
</tr>
<tr>
<td>Password</td>
<td>PWD</td>
<td>string</td>
</tr>
<tr>
<td>Start</td>
<td>Start</td>
<td>string</td>
</tr>
<tr>
<td>Userid</td>
<td>UID</td>
<td>string</td>
</tr>
</tbody>
</table>

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 (5R09) 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 e-mail 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, errmsg)
  VALUES
    (current_remote_user, error_text);
```

259
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 AUTO_INCREMENT,  
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:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pub</td>
<td>Current publisher of the database (lets you know at what database it was inserted)</td>
</tr>
<tr>
<td>remoteuser</td>
<td>Remote user applying the message (lets you know what database it came from)</td>
</tr>
<tr>
<td>errormsg</td>
<td>Error message passed to the Replication_error procedure</td>
</tr>
</tbody>
</table>
Here is a sample insert into the table from the above error:

```sql
INSERT INTO cons.replication_audit
    ( id,
      pub,
      remoteuser,
      errmsg,
      "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.

```bash
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`.

263
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.
CHAPTER 11 Administering SQL Remote for Adaptive Server Anywhere

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:

dbboxup -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
```

<table>
<thead>
<tr>
<th>Old log names different before 5.5.01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior to release 5.5.01 of Adaptive Server Anywhere, the old log files were named <code>consol.100</code>, <code>consol.101</code>, 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.</td>
</tr>
</tbody>
</table>

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.
Transaction log and backup management

Media failure on the database file

Transaction log (intact)  Database file (corrupt)

Restore backed up database file

Transaction log (intact)  Old database file (intact)

Start database with transaction log

Transaction log (intact)  Database file (restored)

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.

268
6 Start the database using the following command line:
   
   dbeng6 -a C: \ RECOVER \ CONSO.L.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 \textit{cons.log} to the directory \texttt{e:\backdir}. The transaction log file is then renamed to \texttt{dateNN.log}, where \texttt{date} is the current date and \texttt{NN} is the next number in sequence, and a new transaction log is started. The directory \texttt{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

\begin{verbatim}
dbremote -c "uid=dba;pwd=sql" C:\DBDIR
\end{verbatim}

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

\section*{To recover from media failure on the C: drive:}

1. Backup the mirrored transaction log \texttt{d:\mirdir\cons.mlg}.
2. Create a temporary directory to perform the recovery in. We will call it \texttt{c:\recover}.
3. Restore the most recent backup of the database file, \texttt{cons.db} to \texttt{c:\recover\cons.db}.
4. Apply the renamed transaction logs in order, as follows
   \begin{verbatim}
   dbeng6 -a C:\DBDIR\date00.LOG C:\RECOVER\CONS.DB
   dbeng6 -a C:\DBDIR\date01.LOG C:\RECOVER\CONS.DB
   \end{verbatim}
5. Copy the current transaction log, \texttt{c:\dbdir\cons.log} to the recovery directory, giving \texttt{c:\recover\cons.log}.
6. Start the database using the following command:
   \begin{verbatim}
   dbeng6 C:\RECOVER\CONS.DB
   \end{verbatim}
7. Shutdown the database server.
8. Backup the recovered database and transaction log from \texttt{c:\recover}.
9. Copy the files from \texttt{c:\recover} to the appropriate production directories.
   - Copy \texttt{c:\recover\cons.db} to \texttt{c:\dbdir\cons.db}.
   - Copy \texttt{c:\recover\cons.log} to \texttt{c:\dbdir\cons.log}, and to \texttt{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.

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 PASSTHRU statement. Any statement entered between the starting PASSTHRU statement and the PASSTHRU 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:

```sql
PASSTHRU 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:

```sql
PASSTHRU 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.

```sql
PASSTHRU ONLY FOR user_1 ;
statement_1 ;
PASSTHRU ONLY FOR user_2 ;
statement_2 ;
```

The following statement terminates a passthrough session:

```sql
PASSTHRU STOP ;
```

PASSTHRU 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.
Carrying out some of the SQL statements on the node where the SQL Remote setup is running is equivalent to the SQL statements being run on the remote server, because the SQL Remote setup is the default way to execute SQL statements. You can:

- 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:

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

However, the following dynamic SQL statement is replicated:

```sql
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.
Using passthrough mode
CHAPTER 12

Administering SQL Remote for Adaptive Server Enterprise

About this chapter

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

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How the Message Agent for Adaptive Server Enterprise works</td>
<td>278</td>
</tr>
<tr>
<td>Running the Message Agent</td>
<td>283</td>
</tr>
<tr>
<td>Error reporting and handling</td>
<td>285</td>
</tr>
<tr>
<td>Adaptive Server Enterprise transaction log and backup management</td>
<td>286</td>
</tr>
<tr>
<td>Making schema changes</td>
<td>289</td>
</tr>
<tr>
<td>Using passthrough mode</td>
<td>290</td>
</tr>
</tbody>
</table>
How the Message Agent for Adaptive Server Enterprise works

This section describes how the Message Agent for Adaptive Server Enterprise works. There are some significant differences between how the Message Agent for Adaptive Server Enterprise and the Message Agent for Adaptive Server Anywhere operate, which accommodate the different roles of the two servers.

For information on features of the Message Agent that are common to Adaptive Server Anywhere and Adaptive Server Enterprise, see "Running the Message Agent" on page 239.

Message Agent is ssremote

The Message Agent for Adaptive Server Enterprise is the following executable:

- On Windows operating systems, the Message Agent is ssremote.exe
- On UNIX operating systems, the Message Agent is ssremote.

Scanning the transaction log

The Message Agent scans the Adaptive Server Enterprise transaction log in order to collect transactions to be sent to remote databases. It stores these transactions in a stable queue.

For more information about the stable queue, see "The stable queue" on page 279. For more information about how the Message Agent uses the stable queue, see "Message Agent operation phases" on page 280.

The SQL Remote Message Agent uses the same transaction log scanning interface as the Adaptive Server Enterprise Log Transfer manager (LTM). Adaptive Server Enterprise maintains a truncation point, which is an identifier for the oldest page in the transaction log needed by the replication system.

The SQL Remote Message Agent sets the truncation point as soon as transactions are scanned from the transaction log and committed in the stable queue. This allows the dump transaction command to reclaim space in the transaction log as soon as possible. The Message Agent does not wait until confirmation is received from remote databases before setting the truncation point.
### Replication Server and SQL Remote

Using SQL Remote on an Adaptive Server Enterprise database participating in a Replication Server setup may involve other considerations. If your database has a replication agent (LTM) running against it, then you need to use the SQL Remote Open Server as an additional component. Adaptive Server Enterprise databases have replication agents running against them in the following circumstances:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>♦</td>
<td>The database is participating in a Replication Server setup as a primary database, or</td>
</tr>
<tr>
<td>♦</td>
<td>The database is participating in a Replication Server setup and is using asynchronous procedure calls.</td>
</tr>
</tbody>
</table>

If the database is participating in a Replication Server setup as a replicate site, and no asynchronous procedure calls are being used, there is no need for the SQL Remote Open Server.

For more information about the SQL Remote Open Server, see "Using SQL Remote with Replication Server" on page 291.

### The stable queue

The Message Agent for Adaptive Server Enterprise uses a **stable queue** to hold transactions until they can be deleted. A stable queue is a pair of database tables that hold messages that may still be needed by the replication system.

SQL Remote for Adaptive Server Anywhere does not use a stable queue.

### Stable queue not identical to Replication Server stable queue

Sybase Replication Server also uses stable queues as storage areas for replication messages. The Replication Server and SQL Remote stable queues perform similar functions, but are *not* the same thing.

### Stable queue location

The stable queue may be kept in the same database as the tables being replicated, or in a different database. Keeping the stable queue in a separate database complicates the backup and recovery plan, but can improve performance by putting the stable queue workload on separate devices and/or a separate Adaptive Server Enterprise server.
**Do not modify the stable queue directly**

The stable queue is maintained by and for the Message Agent. You should not modify the stable queue directly.

**Stable queue tables**

The stable queue consists of two tables. The *sr_transaction* table has one row for each transaction in the stable queue, and the supplementary table *sr_queue_state* has a single row, which stores persistent global information about the state of the stable queue.

_for a description of each of the columns of these tables, see "Stable Queue tables" on page 347.

**Message Agent operation phases**

The Message Agent has the following phases of execution:

- **Receiving messages** During this phase, the Message Agent receives incoming messages and applies them to the Adaptive Server Enterprise server.

- **Populating the stable queue** During this phase the Message Agent scans the Adaptive Server Enterprise transaction log into the stable queue.
- **Sending messages**  During this phase, the Message Agent builds outgoing messages from the stable queue.
The transactions remain in the stable queue until confirmation has been received from all remote databases. When confirmation is received, the transactions are automatically removed from the stable queue by the Message Agent.

The Message Agent does not scan the transaction log for the stable queue.

For information on running multiple copies of the Message Agent to carry out these tasks, see "Running multiple Message Agents" on page 283.
Running the Message Agent

This section describes how to run the Message Agent for Adaptive Server Enterprise. For information on features of the Message Agent that are common to Adaptive Server Anywhere and Adaptive Server Enterprise, see "Running the Message Agent" on page 239.

The Message Agent and replication security

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

In practice, you will not use such a user ID, but the Message Agent needs to run using a user ID with replication role. You can grant replication role with the following statement:

```
sp_role 'grant', replication_role, user_name
```

Running multiple Message Agents

The three phases of Message Agent operation are described in the section "Message Agent operation phases" on page 280. To summarize, these phases are:

- Receiving messages.
- Scanning the transaction log.
- Sending messages.

You may wish to run separate copies of the Message Agent to carry out these different phases. You can specify which phases a given Message Agent is to execute on the Message Agent command line.

The command-line options are as follows:

- **Receive** The `-r` command-line switch instructs the Message Agent to receive messages while it is running. To cause the Message Agent to shut down after receiving available messages, use the `-b` switch in addition to `-r`.

- **Scan log** The `-i` command-line switch instructs the Message Agent to scan the transaction log into the stable queue while it is running.
Running the Message Agent

- **Send** The -s command-line switch instructs the Message Agent to send messages while it is running.

- **Multiple phases** If none of -r, -i, or -s is specified, the Message Agent executes all three phases. Otherwise, only the indicated phases are executed.

There are several circumstances where you may wish to run multiple Message Agents.

- **Ensuring the transaction log does not run out of space** It is important that the transaction log not be allowed to become full. For this reason, you must scan the transaction log frequently enough to ensure that all entries required by SQL Remote are placed in the stable queue. Therefore, you may want to run a Message Agent that scans the transaction log continuously, even if you are only receiving and sending messages in batch mode.

- **Mixing operating systems** If you wish to use a message link supported under Windows 95 or NT, you must use a Message Agent on that platform to send and receive messages. You can do this, while running the log scanning on a UNIX machine, by running two copies of the Message Agent.

How Message Agents are synchronized

The operations of two or more Message Agents are synchronized by a table called `sr_marker`. This table has a single column called `marker`, of data type `datetime`.

When the Message Agent wants to wait for transactions to be scanned into the stable queue, it updates `sr_marker` and waits for it to work its way through the system. The column in `sr_queue_state` is also called marker, and contains the most recent marker to be scanned from the transaction log.
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.

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 executed by the Message Agent when an error occurs. By default no procedure is executed.

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.

285
Adaptive Server Enterprise transaction log and backup management

You must protect against losing transactions that have been replicated to remote databases. If transactions are lost that have already been replicated to remote databases, the remote databases will be inconsistent with the consolidated database. In this situation, you may have to re-extract all remote databases.

Protecting against media failure on the transaction log

Media failure on the transaction log can cause committed transactions to be lost. If the transaction log has been scanned and these transactions have already been sent to subscriber databases, then the subscribing databases contain transactions that are lost from the publishing database, and the databases are in an inconsistent state.

Why the transaction log is needed

The transaction log is needed, even after the entries have been scanned into the stable queue, to guard against media failure on the database file. If the database is lost, it must be recovered to a point that includes every transaction that may have been sent to remote databases.

This recovery is done by restoring a database dump and loading transaction dumps to bring the database up to date. The last transaction dump restored is the dump of the active transaction log at the time of the failure.

Protecting against transaction log loss

There are two ways of protecting against inconsistency arising from media failure on the transaction log:

- **Mirror the transaction log** When a device is mirrored, all writes to the device are copied to a separate physical device.

- **Only replicate backed-up transactions** There is a command-line switch for the Message Agent that prevents it from sending transactions until they are backed up.

Mirroring the transaction log

The only way to protect against media failure on the transaction log is by mirroring the transaction log.

Disk mirroring can provide nonstop recovery in the event of media failure. The disk mirror command causes an Adaptive Server Enterprise database device to be duplicated—that is, all writes to the device are copied to a separate physical device. If one of the devices fails, the other contains an up-to-date copy of all transactions.
Chapter 12  Administering SQL Remote for Adaptive Server Enterprise

Replicating only backed-up transactions

The Message Agent also provides a command line switch (-u) that only sends transactions that have been backup up. In Adaptive Server Enterprise, this means transactions complete before the latest dump database command or dump transaction command.

Choosing an approach

The goal of the strategy is to reduce the possibility of requiring re-extraction of remote databases to an acceptable level. In large setups, the possibility must be as close to zero as possible, as the cost of re-extraction (in terms of down time) is very high.

- The Message Agent -u command-line switch can be used instead of transaction log mirroring when recovery of all transactions in a consolidated database is not needed and mirroring is considered too expensive. This may be true in small setups or setups where there are no local users on the consolidated database.

- The Message Agent -u command-line switch can also be used in addition to mirroring to provide additional protection against total site failure or double media failure.

Stable queue recovery issues

Keeping the stable queue in a separate database complicates backup and recovery, as consistent versions of the two databases have to be recovered.

Normal recovery automatically restores the two databases to a consistent state, but recovery from media failure takes some care. When restoring database dumps and transaction dumps, it is important to recover the stable queue to a consistent point.

Two procedures in the stable store database are provided to help with recovery from media failure:

- **sp_queue_dump_database**  This procedure is called whenever a dump database is scanned from the transaction log.

- **sp_queue_dump_transaction**  This procedure is called whenever a dump transaction is scanned from the transaction log.

You can modify these stored procedures to issue dump database and dump transaction commands in the stable store database.
Transaction log management

The Adaptive Server Enterprise log transfer interface allows the Message Agent to scan the Adaptive Server Enterprise transaction log. When this interface is being used, it sets a truncation point in the transaction log. The truncation point prevents Adaptive Server Enterprise from re-using pages in the transaction log before they have been scanned by SSREMOTE. For this reason, DUMP TRANSACTION will not necessarily release transaction log pages that are before the oldest open transaction. DUMP TRANSACTION will not release transaction log pages beyond the "truncation point".

Initializing the truncation point

The SQL Remote setup script (ssremote.sql) initializes the truncation point with the following command

```sql
dbcc settrunc( 'ltm', 'valid' ).
```

The truncation point can be reset with the following command

```sql
dbcc settrunc( 'ltm', 'ignore' ).
```

This command tells Adaptive Server Enterprise to ignore the truncation point, allowing transaction log pages beyond the truncation point to be released for reuse. You should only use this command when you are no longer interested in SQL Remote replication with the database and you want to be able to reclaim space in the transaction device with DUMP TRANSACTION commands. Continuing to run SQL Remote after ignoring the truncation point will fail to replicate any transactions that were in transaction log pages that were not scanned by the Message Agent and were freed by DUMP TRANSACTION.
Making schema changes

Schema changes to tables being replicated by SQL Remote must be made on a quiet system. A quiet system means the following:

♦ **No transactions being replicated** There can be no transactions being replicated that modify the tables that are to be altered. All transactions that modify tables being altered must be scanned from the transaction log into the stable queue before the schema is altered. This is performed by running the Message Agent normally, or using the \(-i \ -b\) switches. After the Message Agent completes, you can make the schema change.

♦ **Message Agent** The Message Agent must be shut down when the schema change is being made.

♦ **SQL Remote Open Server** If you are using the SQL Remote Open Server, it must be shut down when the schema change is being made.

Schema changes include changes to publications, such as adding articles or modifying articles. However, creating or dropping subscriptions, and adding or removing remote users do not need to be done on a quiet system.

In the Adaptive Server Enterprise transaction log, there is no information recording table structure changes: the SQL Remote log scanning process gets the table structure from the Adaptive Server Enterprise system tables. Consequently, the Message Agent cannot scan an operation from the transaction log that happened against the old table structure.

Information stored in the stable queue before the schema change uses the old table definitions and information stored after the schema change uses the new table definitions.

Passthrough mode can be used at the same time as the schema change to make sure that schema changes at remote databases occur in the correct sequence.
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.

Determining recipients of passthrough statements

Passthrough destinations are determined by sp_passthrough_user and sp_passthroughSubscription. Executing either of these procedures determines a set of recipients for any subsequent passthrough statements.

Executing either sp_passthrough_user and sp_passthroughSubscription adds to the current list of recipients. The sp_passthrough_stop procedure resets passthrough (that is, resets the list of recipients to be empty).

In Adaptive Server Enterprise, sp_passthrough never executes statements in the Adaptive Server Enterprise server.

Passthrough statements

To cause passthrough SQL statements to replicate, you call sp_passthrough. Due to the VARCHAR(255) limitation in Adaptive Server Enterprise, you should build a long SQL statement up in pieces. Calls to sp_passthrough_piece will build up a single SQL statement. Calling sp_passthrough with the last piece will cause the built up statement to replicate.

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.

Schema modifications

The Adaptive Server Enterprise log transfer interface does not contain information about the number of columns and data types of the columns in a table. SSREMOTE gets this information directly from the Adaptive Server Enterprise system tables. For this reason, altering a table and then scanning operations that happened before the ALTER TABLE will lead to errors. SSREMOTE must set the "truncation point" beyond all operations on replicated tables before schema changes can be made. Operations on replicated tables need to be prevented between SSREMOTE running and the schema changes being made.
CHAPTER 13

Using SQL Remote with Replication Server

About this chapter

This chapter describes the additional components needed to use SQL Remote on an Adaptive Server Enterprise database that also participates in a Replication Server installation.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>When you need to use the SQL Remote Open Server</td>
<td>292</td>
</tr>
<tr>
<td>Architecture for Replication Server/SQL Remote installations</td>
<td>293</td>
</tr>
<tr>
<td>Setting up SQL Remote Open Server</td>
<td>296</td>
</tr>
<tr>
<td>Configuring Replication Server</td>
<td>299</td>
</tr>
<tr>
<td>Other issues</td>
<td>302</td>
</tr>
</tbody>
</table>
When you need to use the SQL Remote Open Server

The Message Agent for Adaptive Server Enterprise scans the Adaptive Server Enterprise transaction log to populate the stable queue, as described in the section "The stable queue" on page 279). SQL Remote messages are built from the transactions in the stable queue.

The Message Agent uses the same interface to scan the transaction log as the Replication Agent for Adaptive Server Enterprise. This means the Message Agent cannot scan the transaction log of an Enterprise database that is a primary site in a Replication Server setup (or a replicate site that allows asynchronous updates to primary data).

If there is a Replication Agent running against your Adaptive Server Enterprise database, you must use the SQL Remote Open Server as an additional component. In this case, SQL Remote is set up so that Replication Server populates the stable queue. The SQL Remote Message Agent does not scan the transaction log. Instead, the SQL Remote Open Server receives transactions from Replication Server. The transactions are parsed by the SQL Remote Open Server and stored in the SQL Remote stable queue.

 água  This chapter assumes knowledge of Replication Server. For information, see your Replication Server documentation.

<table>
<thead>
<tr>
<th>Open Server runtime components required</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Open Server runtime components are not included with SQL Remote. You must obtain them separately from Sybase in order to use the SQL Remote Open Server.</td>
</tr>
</tbody>
</table>
Architecture for Replication Server/SQL Remote installations

The arrangement for using a database as a Replication Server primary site and as a SQL Remote database is illustrated in the following diagram. The diagram illustrates a case where the stable queue is held in the a different database from the data being replicated. The stable queue may alternatively be held in the same database as the data being replicated. All connections are client/server connections, and so the components may be running on the same or different machines.

How the pieces fit together

The SQL Remote Open Server acts as a replicate database in the Replication Server setup, and so replication definitions and subscriptions are required in the Adaptive Server Enterprise database on all tables participating in SQL Remote replication and on several of the SQL Remote system tables.
All operations are replicated to the SQL Remote Open Server, which stores them in the stable queue. The stable queue does not have copies of the tables being replicated. It parses the inserts, updates, and deletes to build transactions. All transactions are stored in an image column of a single table. These transactions are used by the Message Agent to build SQL Remote messages.

The Message Agent always applies incoming SQL Remote messages directly to Adaptive Server Enterprise. It does not send operations to Replication Server. Incoming messages are applied directly to the consolidated database regardless of how the stable queue is populated. Conflict resolution is also performed in the same way.
Chapter 13 Using SQL Remote with Replication Server

Replication Server and SQL Remote

SQL Remote allows two-way replication between the consolidated database and remote databases. Replication Server is performing one-way replication from the consolidated database to the SQL Remote Open Server. From Replication Server’s perspective, transactions that originate in remote SQL Remote databases appear as transactions originating in the consolidated SQL Remote database.

SQL Remote system tables

The SQL Remote Open Server requires information from the SQL Remote system tables concerning publications and subscriptions. The Open Server uses a connection to the Adaptive Server Enterprise database holding that information to retrieve it when it starts.

If the SQL Remote system tables are updated while the Open Server is running, the SQL Remote Open Server needs to receive this information at the correct time. For this reason, some of the SQL Remote system tables need to be marked for replication. This is described in "Setting up SQL Remote Open Server" on page 296.

The SQL Remote Open Server executable

The SQL Remote Open Server is the following executable:

- On Windows operating systems, the SQL Remote Open Server is ssqueue.exe.
- On UNIX operating systems, the SQL Remote Open Server is ssqueue.
Setting up SQL Remote Open Server

This section describes how to set up a SQL Remote installation using the SQL Remote Open Server. The procedure depends on whether the SQL Remote stable queue is being kept in a separate Adaptive Server Enterprise database from the tables being replicated or in the same Adaptive Server Enterprise database.

For more information about stable queue location, see "The stable queue" on page 279.

The setup procedure assumes you are using the extraction utility to produce an initial copy of the data in each remote database. You must be sure not to use the Replication Server materialization feature for this purpose.

The procedure for setting up SQL Remote Open Server has two stages:

- **Prepare a SQL Remote setup**  This stage depends on whether you have an existing SQL Remote installation or not.
- **Add the SQL Remote Open Server to the setup**  This stage is the same regardless of previous installations.

**To prepare your SQL Remote setup, if you have an existing SQL Remote installation:**

1. On a quiet primary database, use the Message Agent to scan any remaining transactions into the stable queue.

   A quiet database is one where neither the Message Agent not the SQL Remote Open Server is running, and where no transactions are being replicated.

2. Follow the steps in the section "Upgrading SQL Remote for Adaptive Server Enterprise" on page 36 to upgrade your SQL Remote software at the consolidated site.

3. Invalidate the Message Agent truncation point at the consolidated database using the following command:

   ```
   dbcc settrunc('ltm', 'ignore')
   ```

4. At the stable queue database, execute the stored procedure `sp_queue_log_transfer_reset`.

**To prepare your SQL Remote setup, with no existing installation:**

1. Set up SQL Remote as described in "Setting Up SQL Remote" on page 29.
2 Set up your SQL Remote publications and subscriptions at this point. For information on this procedure, see "SQL Remote Design for Adaptive Server Enterprise" on page 159.

3 Extract the remote databases. For information on this procedure, see "Using the extraction utility" on page 209.

You are now ready to set up the SQL Remote Open Server.

To set up the SQL Remote Open Server:

1 If the SQL Remote stable queue is in a separate database:
   ♦ Set up the stable queue database as a replicate database in a Replication Server setup. This will create the tables and procedures needed by Replication Server, such as rs_lastcommit.
   ♦ Drop the Replication Server connection to the stable queue database.

2 Add an entry to your interfaces file for the SQL Remote Open Server. The default name used on the SQL Remote Open Server command line is SSQueue.

3 Start the SQL Remote Open Server.

4 Create a Replication Server connection to the SQL Remote Open Server. The user ID and password for this connection must match the user ID and password specified on the SQL Remote Open Server command line for the stable queue connection (that is, the −cq switch, or −c if −cq is not specified).

Configure Replication Server now
You should configure Replication Server for this connection at this point. For a description, see "Configuring Replication Server" on page 299.

5 Define, activate, and validate Replication Server replication definitions and subscriptions for the SQL Remote tables sr_marker, sr_remoteuser, sr_subscription, and sr_passthrough. The script ssremote.rs is a sample script to perform this task. You will need to edit the server and database names in the script to match your names.

If the SQL Remote system tables have any data in them, create the replication definitions so that no materialization happens.
For information on creating replication definitions with no materialization, see the *Replication Server Administration Guide*. The section in *Chapter 7, Managing Subscriptions* entitled Bulk Materialization describes how to set up Replication Server for the case where data exists at a remote database.

6 Define, activate, and validate replication definitions and subscriptions for the tables in your database that need to be replicated by SQL Remote. These must be created without materialization.
Chapter 13  Using SQL Remote with Replication Server

Configuring Replication Server

This section describes how to configure Replication Server for use with the SQL Remote Open Server.

The Replication Server connection to the SQL Remote Open Server must have several configuration parameters set.

Set the dsi_xact_group_size parameter

By default, Replication Server groups multiple transactions into larger transactions. The dsi_xact_group_size parameter controls the maximum size of a grouped transaction.

The dsi_xact_group_size parameter must be set to –1 to disable transaction grouping. Transactions that originate from different remote databases in a SQL Remote setup must not be grouped together.

How to set the parameter

You can set the parameter using the following statement:

```sql
CONFIGURE CONNECTION TO  "ssqueue_server"
SET dsi_xact_group_size TO '-1'
```

Set the dsi_fadeout_time parameter

For performance reasons, the SQL Remote Open Server does not commit changes to the stable queue after every transaction. The SQL Remote Open Server does commit changes to the stable queue when any of the following happens:

♦ One thousand transactions are sent to the SQL Remote Open Server
♦ Replication Server disconnects from the SQL Remote Open Server
♦ An operation on sr_marker replicates through to the SQL Remote Open Server

The commits to the stable queue are needed for the following reasons:

♦ Make newly replicated transactions available to the Message Agent sending process
♦ Release any page locks in the stable queue
Configuring Replication Server

The dsi\_fadeout\_time connection parameter controls the amount of idle
time before Replication Server disconnects from a data server. This should
be set low so the SQL Remote Open Server will commit changes to the
stable queue whenever there is idle time. A value of 20 seconds is
recommended.

How to set the parameter

You can set the parameter using the following statement:

```
CONFIGURE CONNECTION TO "ssqueue_server"
SET dsi\_fadeout\_time TO '20'
```

Set the dsi\_num\_threads parameter

The SQL Remote Open Server does not support multiple DSI threads.
Replication Server should not be configured to use multiple DSI threads on
SQL Remote connections.

Create replication definitions for SQL Remote data

Replication definitions for tables being replicated by SQL Remote must have
certain characteristics. This section describes those characteristics.

In some circumstances SQL Remote replicates an UPDATE operation as an
INSERT or a DELETE (see "Replication of updates" on page 101). This is
referred to as subscription migration in the Replication Server
documentation. In order to replicate an UPDATE as an INSERT, SQL
Remote requires the full pre-image of the row. This means that Replication
Server must specify the values of every column in the WHERE clause of any
UPDATE to a table that might need to be replicated as an INSERT.

The simplest way to achieve this is to list all columns in the PRIMARY KEY
of the replication definition. This forces Replication Server to include every
column in the WHERE clause of every UPDATE. REPlicate MINIMAL
COLUMNS can be used on these replication definitions to prevent every
column from being listed in the SET clause of the UPDATE.

Using the dsi\_data\_style data style

Replication Server 11.5 has a new dsi\_data\_style for SQL Remote. This
data style automatically includes all columns in the WHERE clause of every
UPDATE. It is not necessary to list all columns in the PRIMARY KEY of
the replication definition. A replication definition using REPlicate
MINIMAL COLUMNS prevents Replication Server from keeping the full
pre-image of rows being updated, so the SQL Remote dsi\_data\_style will
not work with REPlicate MINIMAL COLUMNS.
Suspend and restart the connection

After configuring the Replication Server connection to the SQL Remote Open Server, you should suspend and resume the connection so that the parameter settings can take effect. The following commands accomplish this task:

```sql
suspend connection to ssqueue_name
go
resume connection to ssqueue_name
go
```
Other issues

This section lists other issues regarding using SQL Remote with Replication Server.

Running the Message Agent The Message Agent should be run with command line switches to receive and send (-r and -s). This will prevent the Message Agent from attempting to scan the transaction log. If the Message Agent attempts to scan the transaction log while the Replication Agent is running, it will get an error attempting to reserve the "log transfer context".

Procedure calls in SQL Remote Open Server The SQL Remote Open Server passes all procedure calls it receives from Replication Server through to the stable queue database. For example, `rs_get_lastcommit` and `rs_update_lastcommit` are executed in the stable queue database.

Coordinated dumps Replication Server provides a mechanism to coordinate database dumps and transaction log dumps between the main database and the stable queue database. The `rs_dumpdb` and `rs_dumptran` function strings can be used to perform coordinated dumps of the stable queue database. Please see the Replication Server documentation for more information.

Schema changes If you make any schema changes to a SQL Remote installation, you must do so on a quiet system. This includes shutting down the SQL Remote Open Server.
This part presents reference material for SQL Remote.
Utilities and Options Reference

About this chapter

This chapter provides reference material for the SQL Remote command line utilities and SQL Remote database options.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Message Agent</td>
<td>306</td>
</tr>
<tr>
<td>The Database Extraction utility</td>
<td>314</td>
</tr>
<tr>
<td>The SQL Remote Open Server</td>
<td>321</td>
</tr>
<tr>
<td>SQL Remote options</td>
<td>323</td>
</tr>
</tbody>
</table>
The Message Agent

Purpose

To send and apply SQL Remote messages, and to maintain the message tracking system to ensure message delivery.

Syntax

```
{ dbremote | ssremote } [ switches ] [ directory ]
```

Command-line switches

<table>
<thead>
<tr>
<th>Switch</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@filename</td>
<td>Read in switches from configuration file</td>
</tr>
<tr>
<td>@envvar</td>
<td>Read in switches from environment variable</td>
</tr>
<tr>
<td>-a</td>
<td>Do not apply received transactions</td>
</tr>
<tr>
<td>-b</td>
<td>Run in batch mode</td>
</tr>
<tr>
<td>-c &quot;keyword=value; ...&quot;</td>
<td>Supply database connection parameters</td>
</tr>
<tr>
<td>-cq &quot;keyword=value; ...&quot;</td>
<td>Supply database connection parameters for the stable queue (Adaptive Server Enterprise only)</td>
</tr>
<tr>
<td>-dl</td>
<td>Display log messages on screen</td>
</tr>
<tr>
<td>-e locale-string</td>
<td>Locale setting (Adaptive Server Enterprise only)</td>
</tr>
<tr>
<td>-fq</td>
<td>Full scan of the stable queue when sending messages (Adaptive Server Enterprise only)</td>
</tr>
<tr>
<td>-g n</td>
<td>Group transactions consisting of less than n operations.</td>
</tr>
<tr>
<td>-i</td>
<td>Scan transactions from the transaction log into the stable queue (Adaptive Server Enterprise only).</td>
</tr>
<tr>
<td>-k</td>
<td>Close window on completion</td>
</tr>
<tr>
<td>-l length</td>
<td>Maximum message length</td>
</tr>
<tr>
<td>-m size</td>
<td>Maximum amount of memory used for building messages.</td>
</tr>
<tr>
<td>-o file</td>
<td>Output messages to file</td>
</tr>
<tr>
<td>-ot file</td>
<td>Truncate file and log output messages</td>
</tr>
<tr>
<td>-p</td>
<td>Do not purge messages</td>
</tr>
<tr>
<td>-q</td>
<td>Run with minimized window</td>
</tr>
<tr>
<td>-r</td>
<td>Receive messages</td>
</tr>
<tr>
<td>-rd minutes</td>
<td>Polling frequency for incoming messages</td>
</tr>
<tr>
<td>-rp number</td>
<td>Number of receive polls before message is assumed lost</td>
</tr>
<tr>
<td>-ru time</td>
<td>Waiting period to re-scan log on receipt of a resend.</td>
</tr>
<tr>
<td>-s</td>
<td>Send messages</td>
</tr>
</tbody>
</table>
-sd time | Send polling period
-t | Replicate all triggers (Adaptive Server Anywhere only)
-u | Process only backed up transactions
-v | Verbose operation
-w n | Number of worker threads to apply incoming messages (Windows NT and Solaris only)
-x | Rename and restart the transaction log (Adaptive Server Anywhere only).

directory | The directory in which old transaction logs are held (Adaptive Server Anywhere only)

**Description**

The Message Agent sends and applies messages for SQL Remote replication, and maintains the message tracking system to ensure message delivery.

The name of the Message Agent executable is as follows:

- **dbremote** The Message Agent for Adaptive Server Anywhere.
- **ssremote** The Message Agent for Adaptive Server Enterprise.

For Adaptive Server Anywhere, the user ID in the Message Agent command line must have either REMOTE DBA or DBA authority. For Adaptive Server Enterprise, the user ID must have replication role.

The optional *directory* parameter specifies a directory in which old transaction logs are held, so that the Message Agent has access to events from before the current log was started.

The Message Agent uses a number of connections to the database. For a listing, see "Connections used by the Message Agent" on page 240.

For information on REMOTE DBA authority, see "The Message Agent and replication security" on page 257.

**Command-line switch details**

**@filename** Read in command-line switches from the supplied file.

The file may contain line breaks, and may contain any set of command line switches. For example, the following command file holds a set of command line switches for a Message Agent that starts with a cache size of 4 Mb, sends messages only, and connects to a database named **field** on a server named **myserver**:

```
-m 4096
-s
-c "eng=myserver;dbn=field;uid=sa;pwd=sysadmin"
```
If this configuration file is saved as `c:\config.txt`, it can be used in an command line as follows:

```
ssremote @c:\config.txt
```

or

```
dbremote @c:\config.txt
```

**@environment-variable**  Read in command-line switches from the supplied environment variable.

The environment variable may contain any set of command line switches. For example, the first of the following pair of statements sets an environment variable holding a set of command line switches for a database server that starts with a cache size of 4 Mb, receives messages only, and connects to a database named **field** on a server named **myserver**. The **set** statement should be entered all on one line:

```
set envvar=-m 4096 -s -c 
"eng=myserver;dbname=field;uid=sa;pwd=sysadmin"
ssremote @envvar
```

-a  Process the received messages (those in the inbox) without applying them to the database. Used together with -v (for verbose output) and -p (so the messages are not purged), this flag can help detect problems with incoming messages. Used without -p, this flag purges the inbox without applying the messages, which may be useful if a subscription is being restarted.

-b  Run in batch mode. In this mode, the Message Agent processes incoming messages, scans the transaction log once and processes outgoing messages, and then stops.

-c "**parameter=value; ...**"  Specify connection parameters. For Adaptive Server Anywhere, if this option is not specified, the environment variable SQLCONNECT is used.

For example, the following statement runs `dbremote` on a database file named `c:\asa6\asademo.db`, connecting with user ID **DBA** and password **SQL**:

```
dbremote -c "uid=dba;pwd=sql;dbf=c:\asa6\asademo.db"
```

The Message Agent must be run by a user with REMOTE DBA authority or DBA authority.

Note: For information on REMOTE DBA authority, see "The Message Agent and replication security" on page 257.
The Message Agent for Adaptive Server Anywhere supports the full range of Adaptive Server Anywhere connection parameters. The Message Agent for Adaptive Server Enterprise supports the following connection parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID</td>
<td>Login ID</td>
</tr>
<tr>
<td>PWD</td>
<td>Password</td>
</tr>
<tr>
<td>DBN</td>
<td>(optional!) Database name. If this parameter is not supplied, the connection defaults to the default database for the login ID.</td>
</tr>
<tr>
<td>ENG</td>
<td>Adaptive Server Enterprise name.</td>
</tr>
</tbody>
</table>

-cq "parameter=value; ..." Specify connection parameters for the stable queue. This option applies to Adaptive Server Enterprise only. If not supplied, the values default to the –c values.

-dl Display messages in the Message Agent window or on the command line and also in the log file.

-e locale-string This option applies to Adaptive Server Enterprise only. Specify Adaptive Server Enterprise locale information. The locale string has the following format:

"language_name,charset_name[,sort_order]"

By default, the Message Agent uses the default locale, which is defined in the file sybase\locales\locales.dat.

If language_name and charset_name are not supplied, the Message Agent obtains them from Adaptive Server Enterprise. If sort_order is not supplied, the Message Agent uses a binary sort order (sort by byte value).

-fq This option is for use only with Adaptive Server Enterprise. It permits a full scan of the stable queue when sending messages, starting from the oldest confirm_sent value.

This feature is intended for occasional use to clean out a large stable queue. If, for example, a single user has not confirmed receipt of a message from a long time ago, the stable queue may be very large. However, by running –fq you can delete entries from more up-to-date users that have been confirmed, even though they are more recent than the cutoff value at which entries are deleted by default.

-g n Instructs the Message Agent to group transactions containing less than n operations together with transactions that follow. The default is twenty operations. Increasing the value of n can speed up processing of incoming messages, by doing less commits. However, it can also cause deadlock and blocking by increasing the size of transactions.
-i  Scan transactions from the transaction log into the stable queue. This option is available for Adaptive Server Enterprise only. It is used when you wish to run a separate copy of the Message Agent for scanning the transaction log and for sending and receiving messages.

If none of -x, -i, or -s is specified, the Message Agent executes all three phases. Otherwise, only the indicated phases are executed.

For more information, see "Running multiple Message Agents" on page 283.

-k  Close window on completion.

-l length  Specifies the maximum length of message to be sent, in bytes. Longer transactions are split into more than one message. The default is 51200 (50K).

<table>
<thead>
<tr>
<th>Caution</th>
</tr>
</thead>
<tbody>
<tr>
<td>The maximum message length must be the same at all sites in an installation.</td>
</tr>
</tbody>
</table>

For platforms with restricted memory allocation, the value must be less than the maximum memory allocation of the operating system. For example, Windows 3.x allows a maximum memory allocation of 64K, and the maximum message length must be less than this.

-m size  Specifies a maximum amount of memory to be used by the Message Agent for building messages and caching incoming messages. The allowed size can be specified as n (in bytes), nK, or nM. The default is 2048K (2M).

When all remote databases are receiving unique subsets of the operations being replicated, a separate message for each remote database is built up concurrently. Only one message is built for a group of remote users that are receiving the same operations. When the memory being used exceeds the -m value, messages are sent before reaching their maximum size (as specified by the -l switch).

When messages arrive, they are stored in memory by the Message Agent until they are applied. This caching of messages prevents rereading of that are out of order messages from the message system, which may lower performance on large installations. When the memory usage specified using the -m switch is exceeded, messages are flushed in a least-recently used fashion.

-o  Append output to a log file. Default is to send output to the screen.
-ot  Truncate the log file and then append output messages to it. Default is to send output to the screen.

-p  Process the messages without purging them.

-q  For Windowing operating systems only, starts the Message Agent with a minimized window.

-r  Receive messages. If none of -r, -i, or -s is specified, the Message Agent executes all three phases. Otherwise, only the indicated phases are executed.

The Message Agent runs in continuous mode if called with -r. To have the Message Agent shut down after receiving messages, use the -bswitch in addition to -r.

-rd time  By default, the Message Agent polls for incoming messages every minute. This option (rd stands for receive delay) allows the polling frequency to be configured, which is useful when polling is expensive. The Message Agent checks for incoming messages once after each send cycle, regardless of the polling frequency, so if a value greater then the SEND EVERY frequency is supplied, the Message Agent checks for incoming messages once after each send cycle.

You can use a suffix of s after the number to indicate seconds, which may be useful if you want frequent polling. For example:

    dbremote -f 30s

polls every thirty seconds.

For more information on polling, see "Tuning incoming message polling" on page 245.

-rp  When running in continuous mode, the Message Agent polls at certain intervals for messages. After polling a set number of times (by default, one), if a message is missing, the Message Agent assumes it has got lost and requests that it be resent. On slow message systems, this can result in many unnecessary resend requests. You can set the number of polls before a resend request is issued using this option, to cut down on the number of resend requests.

For more information on configuring this option, see "Tuning incoming message polling" on page 245.

-ru  Control the resend urgency. This is the time between detection of a resend request and when the Message Agent starts fulfilling the request. Use this switch to help the Message Agent collect resend requests from multiple users before rescanning the log. The time unit can be any of {s = seconds; m = minutes; h = hours; d = days}
Send messages. If none of \(-r\), \(-i\), or \(-s\) is specified, the Message Agent executes all three phases. Otherwise, only the indicated phases are executed.

\textbf{-sd time} \hspace{1em} Control the \textit{send delay}, which is the time to wait between polls for more transaction log data to send.

\textbf{-t} \hspace{1em} All trigger actions are replicated. If you do 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.

\textbf{-u} \hspace{1em} Process only transactions that have been backed up. This switch prevents the Message Agent from processing transactions since the latest backup. Using this switch, outgoing transactions and confirmation of incoming transactions are not sent until they have been backed up.

In Adaptive Server Anywhere, this means only transactions from renamed logs are processed. In Adaptive Server Enterprise, this means that only transactions committed before the latest \texttt{dump database} or \texttt{dump transaction} statement are processed.

\textbf{-v} \hspace{1em} Verbose output. This switch displays the SQL statements contained in the messages to the screen and, if the \texttt{-o} switch is used, to a log file.

\textbf{-w n} \hspace{1em} Windows NT and Solaris only. The number of worker threads used to apply incoming messages. The default is zero, which means all messages are applied by the main (and only) thread. A value of 1 (one) would have one thread receiving messages from the message system and one thread applying messages to the database.

The \texttt{-w} switch makes it possible to increase the throughput of incoming messages with hardware upgrades. Putting the consolidated database on a device that can perform many concurrent operations (a RAID array with a striped logical drive) will improve throughput of incoming messages. Multiple processors in the computer running the Message Agent could also improve throughput of incoming messages.

The \texttt{-w} switch will not improve performance significantly on hardware that cannot perform many concurrent operations.

Incoming messages from a single remote database will never be applied on multiple threads. Messages from a single remote database are always applied serially in the correct order.
-x Rename and restart the transaction log after it has been scanned for outgoing messages. In some circumstances, replicating data to a consolidated database can take the place of backing up remote databases, or renaming the transaction log when the engine is shut down.

SQL Remote uses several registry settings (in Windows 95 and NT) or initialization file settings (Windows 3.x) to control aspects of message link behavior.

The message link control parameters are stored in the following places:

- **Windows 95 and Windows NT** In the registry, at the following location:

  ```plaintext
  \%HKEY_CURRENT_USER\Software\Sybase\SQL Remote
  ```

- **Windows 3.x** In the file SQLANY.INI, in your SQL Remote installation directory.

- **NetWare** You should create a file named `dbremote.ini` in the `sys:\system` directory to hold the FILE system directory setting.

   For a listing of registry settings, see the section for each message system under "Using message types" on page 228.
The Database Extraction utility

You can access the remote database extraction utility in the following ways:

♦ From Sybase Central, for interactive use under Windows 95 or NT.
♦ From the system command line, using the sxextract or dbxtract command-line utilities. This is useful for incorporating into batch or command files.

$sxextract$ is the extraction utility for Adaptive Server Enterprise, $dbxtract$ is the extraction utility for Adaptive Server Anywhere.

By default, the extraction utility runs at isolation level zero. If you are extracting a database from an active server, you should run it at isolation level 3 (see "Extraction utility options" on page 317) to ensure that data in the extracted database is consistent with data on the server. Running at isolation level 3 may hamper others' turnaround time on the server because of the large number of locks required. It is recommended that you run the extraction utility when the server is not busy, or run it against a copy of the database (see "Designing an efficient extraction procedure" on page 211).

Objects owned by $dbo$

The $dbo$ user ID owns a set of Adaptive Server Enterprise-compatible system objects in an Adaptive Server Anywhere database.

For Adaptive Server Anywhere, the extraction utility does not unload the objects created for the $dbo$ user ID during database creation. Changes made to these objects, such as redefining a system procedure, are lost when the data is unloaded. Any objects created by the $dbo$ user ID since the initialization of the database are unloaded by the Extraction utility, and so these objects are preserved.

Adaptive Server Enterprise security issue

If you extract an Adaptive Server Enterprise database, the resulting Adaptive Server Anywhere database has the default $DBA$ user ID present. If you do not want this user ID present in the remote database, you need to explicitly REVOKE CONNECT for this user.

Extracting a remote database in Sybase Central

Running the extraction utility from Sybase Central carries out the following tasks related to creating and synchronizing SQL Remote subscriptions:

♦ Creates a command file to build a remote database containing a copy of the data in a specified publication.
- Creates the necessary SQL Remote objects, such as message types, publisher and remote user IDs, publication and subscription, for the remote database to receive messages from and send messages to the consolidated database.

- Starts the subscription at both the consolidated and remote databases.

- To extract a remote database from a running database:
  1. Connect to the database.
  2. Right-click the database and click Extract Database in the popup menu.
  3. Follow the instructions in the wizard.

- To extract a remote database from a database file or a running database as follows:
  1. Open the Database Utilities folder in the left panel.
  2. Double-click Extract a Database in the right panel.
  3. Follow the instructions in the wizard.

For full information on extracting a remote database in Sybase Central, see the Sybase Central online Help.

The extraction command-line utility

<table>
<thead>
<tr>
<th>Purpose</th>
<th>To extract a remote Adaptive Server Anywhere database from a consolidated Adaptive Server Enterprise or Adaptive Server Anywhere database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>`{ SSXTRACT</td>
</tr>
<tr>
<td>Switch</td>
<td>Description</td>
</tr>
<tr>
<td><code>-an database</code></td>
<td>Creates a database file with the same settings as the database being unloaded and automatically reloads it.</td>
</tr>
<tr>
<td><code>-ac &quot;keyword=value; ...&quot;</code></td>
<td>Connect to the database specified in the connect string to do the reload.</td>
</tr>
<tr>
<td><code>-b</code></td>
<td>Do not start subscriptions</td>
</tr>
<tr>
<td><code>-e &quot;keyword=value; ...&quot;</code></td>
<td>Supply database connection parameters</td>
</tr>
<tr>
<td><code>-d</code></td>
<td>Unload data only</td>
</tr>
<tr>
<td><code>-f</code></td>
<td>Extract fully qualified publications</td>
</tr>
<tr>
<td><code>-l level</code></td>
<td>Perform all extraction operations at specified isolation level</td>
</tr>
<tr>
<td><code>-k</code></td>
<td>Close window on completion</td>
</tr>
</tbody>
</table>
## The Database Extraction utility

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n</td>
<td>Extract schema definition only</td>
</tr>
<tr>
<td>-o file</td>
<td>Output messages to file</td>
</tr>
<tr>
<td>-p character</td>
<td>Escape character</td>
</tr>
<tr>
<td>-q</td>
<td>Operate quietly: do not print messages or show windows</td>
</tr>
<tr>
<td>-r file</td>
<td>Specify name of generated reload Interactive SQL command file (default &quot;reload.sql&quot;)</td>
</tr>
<tr>
<td>-u</td>
<td>Unordered data</td>
</tr>
<tr>
<td>-v</td>
<td>Verbose messages</td>
</tr>
<tr>
<td>-x</td>
<td>Use external table loads</td>
</tr>
<tr>
<td>-xf</td>
<td>Exclude foreign keys</td>
</tr>
<tr>
<td>-xp</td>
<td>Exclude stored procedures</td>
</tr>
<tr>
<td>-xt</td>
<td>Exclude triggers</td>
</tr>
<tr>
<td>-xv</td>
<td>Exclude views</td>
</tr>
<tr>
<td>-y</td>
<td>Overwrite command file without confirmation</td>
</tr>
<tr>
<td>directory</td>
<td>The directory to which the files are written. This is not needed if you use -an or -ac</td>
</tr>
<tr>
<td>subscriber</td>
<td>The subscriber for whom the database is to be extracted.</td>
</tr>
</tbody>
</table>

### Description

`ssxtract` is the extraction utility for Adaptive Server Enterprise. It is run against a Adaptive Server Enterprise and creates a command file for a remote Adaptive Server Anywhere database.

`dbxtract` is the extraction utility for Adaptive Server Anywhere. It is run against an Adaptive Server Anywhere database and creates a command file for a remote Adaptive Server Anywhere database.

There is no extraction utility to create remote Adaptive Server Enterprise databases.

The command line extraction utility creates a command file and a set of associated data files. The command file can be run against a newly-initialized Adaptive Server Anywhere database to create the database objects and load the data for the remote database.

By default, the command file is named `reload.sql`.

### SSXtract notes

Not all Adaptive Server Enterprise objects have corresponding objects in Adaptive Server Anywhere. The `ssxtract` utility has the following limitations:

- **Single database**  All extracted objects must be in a single Adaptive Server Enterprise database.
- **Passwords** The password for the extracted user IDs are the same as the user ID itself.

- **Permissions** The extracted user ID is granted REMOTE DBA authority.

- **Named constraints** These are extracted as Adaptive Server Anywhere CHECK constraints.

- **System tables** The `sp_populate_sql_anywhere` SQL Remote procedure builds a set of Adaptive Server Anywhere system tables in TEMPDB from the Adaptive Server Enterprise system tables. The extracted schema comes from these temporary system tables.

For more information about the command-line switches, see "Extraction utility options" on page 317.

### Extraction utility options

**Create a database for reloading (-an)** You can combine the operations of unloading a database, creating a new database, and loading the data using this option.

For example, the following command (which should be entered all on one line) creates a new database file named `asacopy.db` and copies the schema and data for the `field_user` subscriber of `asademo.db` into it:

```
    dbxtract -c "uid=dba;pwd=sql;dbf=asademo.db" -an
    asacopy.db field_user
```

If you use this option, no copy of the data is created on disk, so you do not specify an unload directory on the command line. This provides greater security for your data, but at some cost for performance.

**Reload the data to an existing database (-ac)** You can combine the operation of unloading a database and reloading the results into an existing database using this option.

For example, the following command (which should be entered all on one line) loads a copy of the data for the `field_user` subscriber into an existing database file named `newdemo.db`:

```
    dbxtract -c "uid=dba;pwd=sql;dbf=asademo.db" -ac
    "uid=dba;pwd=sql;dbf=newdemo.db" field_user
```

If you use this option, no copy of the data is created on disk, so you do not specify an unload directory on the command line. This provides greater security for your data, but at some cost for performance.
Do not start subscriptions automatically (-b) If this option is selected, subscriptions at the consolidated database (for the remote database) and at the remote database (for the consolidated database) must be started explicitly using the START SUBSCRIPTION statement for replication to begin.

Connection parameters (-c) A set of connection parameters, in a string.

- **DBXTRACT connection parameters** The user ID should have DBA authority to ensure that the user has permissions on all the tables in the database.

  For example, the following statement (which should be typed on one line) extracts a database for remote user ID `joe_remote` from the `asademo` database running on the `sample_server` server, connecting as user ID DBA with password SQL. The data is unloaded into the `c:\unload` directory.

  ```shell
  ssxtract -c "eng=sample_server;dbn=asademo;uid=dba;pwd=sql" c:\extract joe_remote
  ```

  If connection parameters are not specified, connection parameters from the SQLCONNECTION environment variable are used, if set.

- **SSXTRACT connection parameters** The following connection parameters are supported:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID</td>
<td>Login ID</td>
</tr>
<tr>
<td>PWD</td>
<td>Password</td>
</tr>
<tr>
<td>DBN</td>
<td>(optional) Database name. If this parameter is not supplied, the connection defaults to the default database for the login ID.</td>
</tr>
<tr>
<td>ENG</td>
<td>Adaptive Server Enterprise name.</td>
</tr>
</tbody>
</table>

  `ssxtract` cannot extract passwords. It sets passwords to be the same as the user ID.

**Unload the data only (-d)** If this option is selected, the schema definition is not unloaded, and publications and subscriptions are not created at the remote database. This option is for use when a remote database already exists with the proper schema, and needs only to be filled with data.

**Extract fully qualified publications (-f)** In most cases, you do not need to extract fully qualified publication definitions for the remote database, since it typically replicates all rows back to the consolidated database anyway.
However, you may want fully qualified publications for multi-tier setups or for setups where the remote database has rows that are not in the consolidated database.

**Perform extraction at a specified isolation level** (-l) The default setting is an isolation level of zero. If you are extracting a database from an active server, you should run it at isolation level 3 (see "Extraction utility options" on page 317) to ensure that data in the extracted database is consistent with data on the server. Increasing the isolation level may result in large numbers of locks being used by the extraction utility, and may restrict database use by other users.

**Unload the schema definition only** (-n) With this definition, none of the data is unloaded. The reload file contains SQL statements to build the database structure only. You can use the SYNCHRONIZE SUBSCRIPTION statement to load the data over the messaging system. Publications, subscriptions, PUBLISH and SUBSCRIBE permissions are part of the schema.

**Output messages to file** (-o) Outputs the messages from the extraction process to a file for later review.

**Escape character** (-p) The default escape character (\) can be replaced by another character using this option.

**Operate quietly** (-q) Display no messages except errors. This option is not available from other environments. This is available only from the command-line utility.

**Reload filename** (-r) The default name for the reload command file is `reload.sql` in the current directory.

**Output the data unordered** (-u) By default the data in each table is ordered by primary key. Unloads are quicker with the -u switch, but loading the data into the remote database is slower.

**Verbose mode** (-v) The name of the table being unloaded and the number of rows unloaded are displayed. The SELECT statement used is also displayed.

**Use external loads** (-x) In the reload script, the default is to use the LOAD TABLE statement to load the data into the database. If you choose to use external loads, the Interactive SQL INPUT statement is used instead. The LOAD TABLE statement is faster than INPUT.

INPUT takes the path of the data files relative to the client, while LOAD TABLE takes the path relative to the server.
Exclude foreign key definitions (-xf) You can use this if the remote database contains a subset of the consolidated database schema, and some foreign key references are not present in the remote database.

Exclude stored procedure (-xp) Do not extract stored procedures from the database.

Exclude triggers (-xt) Do not extract triggers from the database.

Exclude views (-xv) Do not extract views from the database.

Operate without confirming actions (-y) Without this option, you are prompted to confirm the replacement of an existing command file.
The SQL Remote Open Server

**Purpose**

To take replication data from Replication Server and apply it to the SQL Remote stable queue. This utility is needed only for databases participating in both Replication Server (and using a Replication Agent) and SQL Remote replication.

**Syntax**

```
ssqueue [ switches ] [ open-server-name ]
```

**Command-line switches**

<table>
<thead>
<tr>
<th>Switch</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open-server-name</td>
<td>An open server name, which must be declared in the interfaces file (sql.ini).</td>
</tr>
<tr>
<td>-c &quot;keyword=value; ...&quot;</td>
<td>Supply database connection parameters</td>
</tr>
<tr>
<td>-cq &quot;keyword=value; ...&quot;</td>
<td>Supply database connection parameters for the stable queue</td>
</tr>
<tr>
<td>-dl</td>
<td>Display messages in window</td>
</tr>
<tr>
<td>-o file</td>
<td>Output messages to file</td>
</tr>
<tr>
<td>-v</td>
<td>Verbose operation</td>
</tr>
</tbody>
</table>

**Description**

The SQL Remote Open Server is used to enable an Adaptive Server Enterprise database to take part in both SQL Remote replication while acting as a primary site in a Replication Server installation (or a replicate site using asynchronous procedure calls).

The name of the executable is as follows:

- `ssqueue.exe` Windows NT platforms.
- `ssqueue` UNIX platforms.

**Command-line switch details**

The SQL Remote Open Server must connect to the SQL Remote Open Server, which therefore must have an open server name. This open server name is set on the command line, and must correspond to a master and query entry in the interfaces file (sql.ini) on the machine running the SQL Remote Open Server, and to a query entry on the interfaces file of the machine running Replication Server.

The default value for the open server name is `SSQueue`.

- `-c "parameter=value; ..."` Specify connection parameters to the database holding the data being replicated. This connection is required for the SQL Remote Open Server to gain access to the SQL Remote system tables.

The connection parameters must come from the following list:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID</td>
<td>Login ID</td>
</tr>
<tr>
<td>PWD</td>
<td>Password</td>
</tr>
<tr>
<td>DBN</td>
<td>(optional) Database name. If this parameter is not supplied, the connection defaults to the default database for the login ID.</td>
</tr>
<tr>
<td>ENG</td>
<td>Server name.</td>
</tr>
</tbody>
</table>

-cq "parameter=value; ..." Specify connection parameters for the stable queue. If not supplied, the values default to the -c values.

-dl Display messages in the window or on the command line and also in the log file.

-o Append output to a log file. Default is to send output to the screen.

-v Verbose output. This switch displays the SQL statements contained in the messages to the screen and, if the -o switch is used, to a log file.
SQL Remote options

Function
Replication options are database options included to provide control over replication behavior.

Anywhere Syntax
SET [ TEMPORARY ] OPTION
    ... [ userid. | PUBLIC. ] option_name = [ option_value ]

Enterprise syntax:
exec sp_remote_option option-name, option-value

Parameters

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option_name</td>
<td>The name of the option being changed.</td>
</tr>
<tr>
<td>option_value</td>
<td>A string containing the setting for the option.</td>
</tr>
</tbody>
</table>

Description
The following options are available.

<table>
<thead>
<tr>
<th>OPTION</th>
<th>VALUES</th>
<th>DEFAULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob_threshold</td>
<td>Integer, in K</td>
<td>256</td>
</tr>
<tr>
<td>Compression</td>
<td>-1 to 9</td>
<td>6</td>
</tr>
<tr>
<td>Delete_old_logs</td>
<td>ON, OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>Qualify_owners</td>
<td>ON, OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>Quote_all_identifiers</td>
<td>ON, OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>Replication_error</td>
<td>procedure-name</td>
<td>NULL</td>
</tr>
<tr>
<td>Subscribe_by_remote</td>
<td>ON,OFF</td>
<td>ON</td>
</tr>
<tr>
<td>Verify_threshold</td>
<td>integer</td>
<td>256</td>
</tr>
<tr>
<td>Verify_all_columns</td>
<td>ON,OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

These options are used by the Message Agent, and should be set for the user ID specified on the Message Agent command line. They can also be set for general public use.

The options are as follows:

- **Blob_threshold option**  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.

  If you are replicating blobs in an installation with Adaptive Server Enterprise, you must ensure that Blob_threshold is set to a value larger than the largest blob being replicated.
For information on blob replication and Adaptive Server Enterprise, see "Replication of blobs" on page 104.

Compression option Set the level of compression for messages. Values can be from -1 to 9, and have the following meanings:
- -1 Send messages in Version 5 format. Message Agents (both dbremote and ssremote) from previous versions of SQL Remote cannot read messages sent in Version 6 format. You should ensure that COMPRESSION is set to -1 until all Message Agents in your system are upgraded to Version 6.
- 0 No compression.
- 1 to 9 Increasing degrees of compression. Creating messages with high compression can take longer than creating messages with low compression.

Delete_old_logs option This option is used by SQL Remote and by the Adaptive Server Anywhere Replication Agent. The default setting is OFF. When set to ON, the Message Agent (DBREMOTE) deletes each old transaction log when all the changes it contains have been sent and confirmed as received.

Qualify_owners option Controls whether SQL statements being replicated by SQL Remote should use qualified object names. The default in Adaptive Server Anywhere is ON and the default in Adaptive Server Enterprise is OFF.

Qualifying owners in Adaptive Server Enterprise setups is rarely needed because it is common for objects to be owned by dbo. When qualification is not needed in Adaptive Server Anywhere setups, messages will be slightly smaller with the option off.

Quote_all_identifiers option Controls whether SQL statements being replicated by SQL Remote should use quoted identifiers. The default is OFF.

When this option is off, the dbremote quotes identifiers that require quotes by Adaptive Server Anywhere (as it has always done) and ssremote does not quote any identifiers. When the option is on, all identifiers are quoted.

Replication_error option Specifies a stored procedure called by the Message Agent when a SQL error occurs. By default no procedure is called.

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

- **Subscribe_by_remote option**  When set to ON, operations from remote databases on rows with a subscribe by value that is NULL or an empty string assume the remote user is subscribed to the row. When set to OFF, the remote user is assumed not to be subscribed to the row.

  The only limitation of this option is that it will lead to errors if a remote user really does want to INSERT (or UPDATE) a row with a NULL or empty subscription expression (for information held only at the consolidated database). This is reasonably obscure and can be worked around by assigning a subscription value in your installation that belongs to no remote user.

  For more information about this option, see "Using the Subscribe_by_remote option with many-to-many relationships" on page 140, and "Using the Subscribe_by_remote option with many-to-many relationships" on page 180.

- **Verify_threshold option**  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. The default setting is 1000.

  This option keeps the size of SQL Remote messages down, but has the disadvantage that conflicting updates of long values are not detected.

- **Verify_all_columns option**  The default setting is OFF. When set to ON, messages containing updates published by the local database are sent with all column values included, and a conflict in any column triggers a RESOLVE UPDATE trigger at the subscriber database.

  The extraction utility for Adaptive Server Enterprise sets the public option in remote Adaptive Server Anywhere databases to match the setting in the Adaptive Server Enterprise database.

**Examples**

- The following statement sets the Verify_all_columns option to OFF in Adaptive Server Anywhere, for all users:

  ```sql
  SET OPTION PUBLIC.Verify_all_columns = 'OFF'
  ```

- The following statements set the Verify_all_columns option to OFF in Adaptive Server Enterprise:

  ```sql
  exec sp_remote_option Verify_all_columns, 'OFF'
  go
  ```

  In Adaptive Server Enterprise, replication options are used only by SQL Remote.
CHAPTER 15

System Objects for Adaptive Server Anywhere

About this chapter

SQL Remote-specific system information is held in a the Adaptive Server Anywhere catalog. A more comprehensible version of this information is held in a set of system views.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Remote system tables</td>
<td>328</td>
</tr>
<tr>
<td>SQL Remote system views</td>
<td>332</td>
</tr>
</tbody>
</table>
SQL Remote system tables

This section describes the system tables used by SQL Remote to define and manage SQL Remote information.

SYSARTICLE table

Function: Each row describes an article in a SQL Remote publication.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>SMALLINT</td>
<td>The publication of which this article is a part.</td>
</tr>
<tr>
<td>table_id</td>
<td>INT</td>
<td>Each article consists of columns and rows from a single table. This column contains the table ID for this table.</td>
</tr>
<tr>
<td>where_expr</td>
<td>VARCHAR(128)</td>
<td>For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.</td>
</tr>
<tr>
<td>subscribe_by_expr</td>
<td>VARCHAR(128)</td>
<td>For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.</td>
</tr>
</tbody>
</table>

SYSARTICLECOL table

Function: Each row identifies a column in an article, identifying the column, the table it is in, and the publication it is part of.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>SMALLINT</td>
<td>A unique identifier for the publication of which the column is a part.</td>
</tr>
<tr>
<td>table_id</td>
<td>INT</td>
<td>The table to which the column belongs.</td>
</tr>
<tr>
<td>column_id</td>
<td>INT</td>
<td>The column identifier, from the SYSCOLUMN system table.</td>
</tr>
</tbody>
</table>

SYSPUBLICATION table

Function: Each row describes a SQL Remote publication.
Chapter 15  System Objects for Adaptive Server Anywhere

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>publication_name</td>
<td>VARCHAR(128)</td>
<td></td>
</tr>
</tbody>
</table>

SYSREMTYPE table

Function
Each row describes one of the SQL Remote message types, including the publisher address.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_id</td>
<td>SMALLINT</td>
<td>An identification number for the message type.</td>
</tr>
<tr>
<td>type_name</td>
<td>VARCHAR(128)</td>
<td>The message type. There is a separate row for each of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ FILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ MAPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ VIM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ SMTP</td>
</tr>
<tr>
<td>publisher_address</td>
<td>VARCHAR(128)</td>
<td>The publisher's address for the message type type_name.</td>
</tr>
</tbody>
</table>

SYSREMOTEUSER table

Function
Each row describes a user ID with REMOTE permissions (a subscriber), together with the status of SQL Remote messages sent to and from that user.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>SMALLINT</td>
<td>The user ID of the user with REMOTE permissions.</td>
</tr>
<tr>
<td>operation</td>
<td>VARCHAR(10)</td>
<td></td>
</tr>
<tr>
<td>consolidate</td>
<td>CHAR(1)</td>
<td>The column contains either an N to indicate a user granted REMOTE permissions, or a Y to indicate a user granted CONSOLIDATE permissions.</td>
</tr>
<tr>
<td>type_id</td>
<td>SMALLINT</td>
<td>The ID of the message system used to send messages to this user.</td>
</tr>
<tr>
<td>Column</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>address</td>
<td>VARCHAR(128)</td>
<td>The address to which SQL Remote messages are to be sent. The address must be appropriate for the <code>address_type</code>.</td>
</tr>
<tr>
<td>frequency</td>
<td>CHAR(1)</td>
<td>How frequently SQL Remote messages are to be sent.</td>
</tr>
<tr>
<td>send_time</td>
<td>DATETIME</td>
<td>The next time messages are to be sent to this user.</td>
</tr>
<tr>
<td>log_send</td>
<td>NUMERIC(20, 0)</td>
<td>Messages are sent only to subscribers for whom <code>log_send</code> is greater than <code>log_sent</code>.</td>
</tr>
<tr>
<td>time_sent</td>
<td>DATETIME</td>
<td>The time the most recent message was sent to this subscriber.</td>
</tr>
<tr>
<td>log_sent</td>
<td>NUMERIC(20, 0)</td>
<td>The log offset for the most recently sent operation.</td>
</tr>
<tr>
<td>confirm_sent</td>
<td>NUMERIC(20, 0)</td>
<td>The log offset for the most recently confirmed operation from this subscriber.</td>
</tr>
<tr>
<td>send_count</td>
<td>INT</td>
<td>How many SQL Remote messages have been sent.</td>
</tr>
<tr>
<td>resend_count</td>
<td>INT</td>
<td>Counter to ensure messages are applied only once at the subscriber database.</td>
</tr>
<tr>
<td>time_received</td>
<td>DATETIME</td>
<td>The time the most recent message was received from this subscriber.</td>
</tr>
<tr>
<td>log_received</td>
<td>NUMERIC(20, 0)</td>
<td>The log offset in the subscriber's database for the operation most recently received at the current database.</td>
</tr>
<tr>
<td>confirm_received</td>
<td>NUMERIC(20, 0)</td>
<td>The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.</td>
</tr>
<tr>
<td>receive_count</td>
<td>INT</td>
<td>How many messages have been received.</td>
</tr>
<tr>
<td>rereceive_count</td>
<td>INT</td>
<td>Counter to ensure messages are applied only once at the current database.</td>
</tr>
</tbody>
</table>
### SYSSUBSCRIPTION table

Each row describes a subscription from one user ID (which must have REMOTE permissions) to one publication.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>SMALLINT</td>
<td>The identifier for the publication to which the user ID is subscribed.</td>
</tr>
<tr>
<td>user_id</td>
<td>SMALLINT</td>
<td>The user ID that is subscribed to the publication.</td>
</tr>
<tr>
<td>subscribe_by</td>
<td>VARCHAR(128)</td>
<td>For publications with a SUBSCRIBE BY expression, this column holds the matching value for this subscription.</td>
</tr>
<tr>
<td>created</td>
<td>NUMERIC(20, 0)</td>
<td>The offset in the transaction log at which the subscription was created.</td>
</tr>
<tr>
<td>started</td>
<td>NUMERIC(20, 0)</td>
<td>The offset in the transaction log at which the subscription was started.</td>
</tr>
<tr>
<td>operation</td>
<td>VARCHAR(20)</td>
<td></td>
</tr>
</tbody>
</table>
SQL Remote system views

This section describes the database views used by SQL Remote to present
and summarize SQL Remote information.

SYSARTICLES view

<table>
<thead>
<tr>
<th>Function</th>
<th>Each row lists describes an article.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The publication of which this article is a part.</td>
</tr>
<tr>
<td>table_name</td>
<td>Each article consists of columns and rows from a single table. This column contains the name of this table.</td>
</tr>
<tr>
<td>where_expr</td>
<td>For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.</td>
</tr>
<tr>
<td>subscribe_by_expr</td>
<td>For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.</td>
</tr>
</tbody>
</table>

SYSARTICLECOLS view

<table>
<thead>
<tr>
<th>Function</th>
<th>Each row describes a column that appears in an article.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication of which the column is a part.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table to which the column belongs.</td>
</tr>
<tr>
<td>column_name</td>
<td>The column name.</td>
</tr>
</tbody>
</table>

SYSPUBLICATIONS view

<table>
<thead>
<tr>
<th>Function</th>
<th>Lists the names of all publications.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Columns</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication</td>
</tr>
</tbody>
</table>

332
SYSREMOTEUSERS view

Lists information about remote users and their status.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>The user ID of the user with REMOTE permissions.</td>
</tr>
<tr>
<td>consolidate</td>
<td>The column contains either an R to indicate a user granted REMOTE permissions, or a C to indicate a user granted CONSOLIDATE permissions.</td>
</tr>
<tr>
<td>type_name</td>
<td>The name of the message system used to send messages to this user.</td>
</tr>
<tr>
<td>address</td>
<td>The address to which SQL Remote messages are to be sent. The address must be appropriate for the address_type.</td>
</tr>
<tr>
<td>frequency</td>
<td>How frequently SQL Remote messages are to be sent.</td>
</tr>
<tr>
<td>send_time</td>
<td>The next time messages are to be sent to this user.</td>
</tr>
<tr>
<td>next_send</td>
<td>The next time messages are to be sent to this user, in a more comprehensible format.</td>
</tr>
<tr>
<td>log_send</td>
<td>Messages are sent only to subscribers for whom log_send is greater than log_sent.</td>
</tr>
<tr>
<td>time_sent</td>
<td>The time the most recent message was sent to this subscriber.</td>
</tr>
<tr>
<td>log_sent</td>
<td>The log offset for the most recently sent operation.</td>
</tr>
<tr>
<td>confirm_sent</td>
<td>The log offset for the most recently confirmed operation from this subscriber.</td>
</tr>
<tr>
<td>send_count</td>
<td>How many SQL Remote messages have been sent.</td>
</tr>
<tr>
<td>resend_count</td>
<td>Counter to ensure messages are applied only once at the subscriber database.</td>
</tr>
<tr>
<td>time_received</td>
<td>The time the most recent message was received from this subscriber.</td>
</tr>
<tr>
<td>log_received</td>
<td>The log offset in the subscriber's database for the operation most recently received at the current database.</td>
</tr>
<tr>
<td>confirm_received</td>
<td>The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.</td>
</tr>
<tr>
<td>receive_count</td>
<td>How many messages have been received.</td>
</tr>
<tr>
<td>rereceive_count</td>
<td>Counter to ensure messages are applied only once at the current database.</td>
</tr>
</tbody>
</table>
## SYSSUBSCRIPTIONS view

Each row lists information about a subscription.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication to which the user ID is subscribed.</td>
</tr>
<tr>
<td>user_name</td>
<td>The user ID that is subscribed to the publication.</td>
</tr>
<tr>
<td>subscribe_by</td>
<td>For publications with a SUBSCRIBE BY expression, this column holds the matching value for this subscription.</td>
</tr>
<tr>
<td>created</td>
<td>The offset in the transaction log at which the subscription was created.</td>
</tr>
<tr>
<td>started</td>
<td>The offset in the transaction log at which the subscription was started.</td>
</tr>
</tbody>
</table>
CHAPTER 16

System Objects for Adaptive Server Enterprise

About this chapter

SQL Remote-specific system information is held in a set of tables called the SQL Remote system tables. A more comprehensible version of this information is held in a set of views, called the SQL Remote system views.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Remote system tables</td>
<td>336</td>
</tr>
<tr>
<td>SQL Remote system views</td>
<td>343</td>
</tr>
<tr>
<td>Stable Queue tables</td>
<td>347</td>
</tr>
</tbody>
</table>
SQL Remote system tables

This section describes the database tables used by SQL Remote to define and manage SQL Remote information.

Caution

These tables are for use only by SQL Remote. Do not alter these tables or their contents directly.

#remote table

Function

This temporary table is created by the Message Agent to hold the name of the current remote user and of the current publisher. This table exists only in Adaptive Server Enterprise.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_remote_user</td>
<td>VARCHAR(128)</td>
<td>Current remote user (from the Message Agent command line).</td>
</tr>
<tr>
<td>current_publisher</td>
<td>VARCHAR(128)</td>
<td>Current publisher</td>
</tr>
</tbody>
</table>

Description

This is not a system table. When the Message Agent for Adaptive Server Enterprise connects to the server, it holds the value of the current remote user ID and the value of the current publisher in the #remote table. This temporary table is held in TEMPDB.

The values from #remote can be used in conflict resolution procedures.

The CREATE TABLE statement for this table is:

```
CREATE TABLE #remote {
    current_remote_user varchar(128),
    current_publisher varchar(128)
}
```

The table has a single row.

sr_article table

Function

Each row describes an article in a SQL Remote publication.
<table>
<thead>
<tr>
<th>Columns</th>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>publication_id</td>
<td>INT</td>
<td>The publication of which this article is a part.</td>
</tr>
<tr>
<td></td>
<td>table_id</td>
<td>INT</td>
<td>Each article consists of columns and rows from a single table. This column contains the table ID for this table.</td>
</tr>
<tr>
<td></td>
<td>where_expr</td>
<td>VARCHAR(128)</td>
<td>For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.</td>
</tr>
<tr>
<td></td>
<td>subscribe_by_expr</td>
<td>VARCHAR(128)</td>
<td>For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.</td>
</tr>
</tbody>
</table>

**sr_articlecol table**

**Function**

Each row identifies a column in an article, identifying the column, the table it is in, and the publication it is part of.

<table>
<thead>
<tr>
<th>Columns</th>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>publication_id</td>
<td>INT</td>
<td>A unique identifier for the publication of which the column is a part.</td>
</tr>
<tr>
<td></td>
<td>table_id</td>
<td>INT</td>
<td>The table to which the column belongs.</td>
</tr>
<tr>
<td></td>
<td>column_id</td>
<td>INT</td>
<td>The column identifier, from the SYSCOLUMN system table.</td>
</tr>
</tbody>
</table>

**sr_marker table**

**Function**

To ensure that messages received by the Message Agent are sent to remote databases in the same session

<table>
<thead>
<tr>
<th>Columns</th>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>marker</td>
<td>DATETIME</td>
<td>A time value indicating when the latest messages were applied.</td>
</tr>
</tbody>
</table>
**Description**

When a consolidated database uses two Message Agents, one to populate the stable queue (-i) and one to receive and send messages (-r -s), the single row of the **sr_marker** table is used to ensure that messages received and applied to the database are sent before the Message Agent closes down.

**sr_object table**

**Function**

Holds a list of SQL Remote objects. The extraction utility needs to know not to extract the SQL Remote system objects. The **sp_populate_sql_anywhere** procedure that creates a set of Adaptive Server Anywhere system tables in TEMPDB gets a list of SQL Remote objects from the **sr_object** table.

**Columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>The name of the object.</td>
</tr>
<tr>
<td>type</td>
<td>CHAR(1)</td>
<td>One of the following: ♦ U User-defined table ♦ V View ♦ P Procedure</td>
</tr>
</tbody>
</table>

**sr_option table**

**Function**

Each row describes a replication option used by SQL Remote.

**Columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>VARCHAR(128)</td>
<td>The name of the option.</td>
</tr>
<tr>
<td>value</td>
<td>VARCHAR(128)</td>
<td>The setting for the option.</td>
</tr>
</tbody>
</table>

**Description**

⚠️ For information about available options, see "SQL Remote options" on page 323.

**sr_passthrough table**

**Function**

Each row describes a passthrough operation being sent to a user or to subscribers to a publication.
### sr_publication_table

**Function**
Each row describes a SQL Remote publication.

**Columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_id</td>
<td>INT</td>
<td>An identifier for the publication</td>
</tr>
<tr>
<td>publication_name</td>
<td>VARCHAR(128)</td>
<td>The name of the publication.</td>
</tr>
</tbody>
</table>

### sr_publisher_table

**Function**
The row holds the user ID of the publisher.

**Columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>INT</td>
<td>The user ID of the publisher.</td>
</tr>
</tbody>
</table>

### sr_remotetable_table

**Function**
Each row describes a table that is marked for replication using SQL Remote.

**Columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_id</td>
<td>INT</td>
<td>The id of the table.</td>
</tr>
<tr>
<td>resolve_name</td>
<td>VARCHAR(128)</td>
<td>The name of the stored procedure to be executed in the case of conflicts.</td>
</tr>
<tr>
<td>old_row_name</td>
<td>VARCHAR(128)</td>
<td>The table that holds the old row name.</td>
</tr>
<tr>
<td>remote_row_name</td>
<td>VARCHAR(128)</td>
<td>The table that holds the remote row name.</td>
</tr>
</tbody>
</table>
### sr_remotetype table

Each row describes one of the SQL Remote message types, including the publisher address.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_id</td>
<td>INT</td>
<td>An identification number for the message type.</td>
</tr>
<tr>
<td>type_name</td>
<td>VARCHAR(128)</td>
<td>The message type. There is a separate row for each of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ FILE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ MAPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ VIM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ SMTP</td>
</tr>
<tr>
<td>publisher_address</td>
<td>VARCHAR(128)</td>
<td>The publisher's address for the message type type_name.</td>
</tr>
</tbody>
</table>

### sr_remoteuser table

Each row describes a user ID with REMOTE permissions (a subscriber), together with the status of SQL Remote messages sent to and from that user.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>INT</td>
<td>The user ID of the user with REMOTE permissions.</td>
</tr>
<tr>
<td>operation</td>
<td>VARCHAR(10)</td>
<td>The column contains either an N to indicate a user granted REMOTE permissions, or a Y to indicate a user granted CONSOLIDATE permissions.</td>
</tr>
<tr>
<td>consolidate</td>
<td>CHAR(1)</td>
<td>The ID of the message system used to send messages to this user.</td>
</tr>
<tr>
<td>type_id</td>
<td>INT</td>
<td>The ID of the message system used to send messages to this user.</td>
</tr>
<tr>
<td>address</td>
<td>VARCHAR(128)</td>
<td>The address to which SQL Remote messages are to be sent. The address must be appropriate for the address_type.</td>
</tr>
<tr>
<td>frequency</td>
<td>CHAR(1)</td>
<td>How frequently SQL Remote messages are to be sent.</td>
</tr>
<tr>
<td>send_time</td>
<td>DATETIME</td>
<td>The next time messages are to be sent to</td>
</tr>
<tr>
<td>Column</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>log_send</td>
<td>NUMERIC(20, 0)</td>
<td>Messages are sent only to subscribers for whom log_send is greater than log_sent.</td>
</tr>
<tr>
<td>time_sent</td>
<td>DATETIME</td>
<td>The time the most recent message was sent to this subscriber.</td>
</tr>
<tr>
<td>log_sent</td>
<td>NUMERIC(20, 0)</td>
<td>The log offset for the most recently sent operation.</td>
</tr>
<tr>
<td>confirm_sent</td>
<td>NUMERIC(20, 0)</td>
<td>The log offset for the most recently confirmed operation from this subscriber.</td>
</tr>
<tr>
<td>send_count</td>
<td>INT</td>
<td>How many SQL Remote messages have been sent.</td>
</tr>
<tr>
<td>resend_count</td>
<td>INT</td>
<td>Counter to ensure messages are applied only once at the subscriber database.</td>
</tr>
<tr>
<td>time_received</td>
<td>DATETIME</td>
<td>The time the most recent message was received from this subscriber.</td>
</tr>
<tr>
<td>log_received</td>
<td>NUMERIC(20, 0)</td>
<td>The log offset in the subscriber's database for the operation most recently received at the current database.</td>
</tr>
<tr>
<td>confirm_received</td>
<td>NUMERIC(20, 0)</td>
<td>The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.</td>
</tr>
<tr>
<td>receive_count</td>
<td>INT</td>
<td>How many messages have been received.</td>
</tr>
<tr>
<td>rereceive_count</td>
<td>INT</td>
<td>Counter to ensure messages are applied only once at the current database.</td>
</tr>
<tr>
<td>filler1</td>
<td>CHAR(255)</td>
<td></td>
</tr>
<tr>
<td>filler2</td>
<td>CHAR(255)</td>
<td></td>
</tr>
<tr>
<td>filler3</td>
<td>CHAR(255)</td>
<td></td>
</tr>
<tr>
<td>filler4</td>
<td>CHAR(255)</td>
<td></td>
</tr>
</tbody>
</table>

**sr_subscription table**

**Function**

Each row describes a subscription from one user ID (which must have REMOTE permissions) to one publication.
## SQL Remote system tables

<table>
<thead>
<tr>
<th>Columns</th>
<th>Column</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>publication_id</td>
<td>INT</td>
<td>The identifier for the publication to which the user ID is subscribed.</td>
</tr>
<tr>
<td></td>
<td>user_id</td>
<td>INT</td>
<td>The user ID that is subscribed to the publication.</td>
</tr>
<tr>
<td></td>
<td>subscribe_by</td>
<td>VARCHAR(128)</td>
<td>For publications with a SUBSCRIBE BY expression, this column holds the matching value for this subscription.</td>
</tr>
<tr>
<td></td>
<td>created</td>
<td>NUMERIC(20, 0)</td>
<td>The offset in the transaction log at which the subscription was created.</td>
</tr>
<tr>
<td></td>
<td>started</td>
<td>NUMERIC(20, 0)</td>
<td>The offset in the transaction log at which the subscription was started.</td>
</tr>
<tr>
<td></td>
<td>operation</td>
<td>VARCHAR(20)</td>
<td></td>
</tr>
</tbody>
</table>
SQL Remote system views

This section describes the database views used by SQL Remote to present and summarize SQL Remote information.

**sr_articles view**

Function  
Each row lists describes an article.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The publication of which this article is a part.</td>
</tr>
<tr>
<td>table_name</td>
<td>Each article consists of columns and rows from a single table. This column contains the name of this table.</td>
</tr>
<tr>
<td>where_expr</td>
<td>For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.</td>
</tr>
<tr>
<td>subscribe_by_expr</td>
<td>For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.</td>
</tr>
</tbody>
</table>

**sr_articlecols view**

Function  
Each row describes a column that appears in an article.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication of which the column is a part.</td>
</tr>
<tr>
<td>table_name</td>
<td>The name of the table to which the column belongs.</td>
</tr>
<tr>
<td>column_name</td>
<td>The column name.</td>
</tr>
</tbody>
</table>

**sr_publications view**

Function  
Lists the names of all publications.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication</td>
</tr>
</tbody>
</table>
**sr_remotetables view**

**Function**
Lists the tables marked for SQL Remote replication, as stored in the remotetable system table, in more readable form.

This table exists only in Adaptive Server Enterprise.

**Columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The name of the table.</td>
</tr>
<tr>
<td>resolve_name</td>
<td>The name of the stored procedure to be executed in the case of conflicts.</td>
</tr>
<tr>
<td>old_row_name</td>
<td>The table that holds the old row name.</td>
</tr>
<tr>
<td>remote_row_name</td>
<td>The table that holds the remote row name.</td>
</tr>
</tbody>
</table>

**sr_remotetypes view**

**Function**
Lists the message types, as stored in the remotetype system table.

**Columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_id</td>
<td>An identification number for the message type.</td>
</tr>
<tr>
<td>type_name</td>
<td>The message type. There is a separate row for each of the following:</td>
</tr>
<tr>
<td></td>
<td>♦ FILE</td>
</tr>
<tr>
<td></td>
<td>♦ MAPI</td>
</tr>
<tr>
<td></td>
<td>♦ VIM</td>
</tr>
<tr>
<td></td>
<td>♦ SMTP</td>
</tr>
<tr>
<td>publisher_address</td>
<td>The publisher's address for the message type type_name.</td>
</tr>
</tbody>
</table>

**sr_remoteusers view**

**Function**
Lists information about remote users and their status.
<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>The user ID of the user with REMOTE permissions.</td>
</tr>
<tr>
<td>consolidate</td>
<td>The column contains either an N to indicate a user granted REMOTE permissions, or a Y to indicate a user granted CONSOLIDATE permissions.</td>
</tr>
<tr>
<td>type_name</td>
<td>The name of the message system used to send messages to this user.</td>
</tr>
<tr>
<td>address</td>
<td>The address to which SQL Remote messages are to be sent. The address must be appropriate for the address_type.</td>
</tr>
<tr>
<td>frequency</td>
<td>How frequently SQL Remote messages are to be sent.</td>
</tr>
<tr>
<td>send_time</td>
<td>The next time messages are to be sent to this user.</td>
</tr>
<tr>
<td>next_send</td>
<td>The next time messages are to be sent to this user, in a more comprehensible format.</td>
</tr>
<tr>
<td>log_send</td>
<td>Messages are sent only to subscribers for whom log_send is greater than log_sent.</td>
</tr>
<tr>
<td>time_sent</td>
<td>The time the most recent message was sent to this subscriber.</td>
</tr>
<tr>
<td>log_sent</td>
<td>The log offset for the most recently sent operation.</td>
</tr>
<tr>
<td>confirm_sent</td>
<td>The log offset for the most recently confirmed operation from this subscriber.</td>
</tr>
<tr>
<td>send_count</td>
<td>How many SQL Remote messages have been sent.</td>
</tr>
<tr>
<td>resend_count</td>
<td>Counter to ensure messages are applied only once at the subscriber database.</td>
</tr>
<tr>
<td>time_received</td>
<td>The time the most recent message was received from this subscriber.</td>
</tr>
<tr>
<td>log_received</td>
<td>The log offset in the subscriber's database for the operation most recently received at the current database.</td>
</tr>
<tr>
<td>confirm_received</td>
<td>The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.</td>
</tr>
<tr>
<td>receive_count</td>
<td>How many messages have been received.</td>
</tr>
<tr>
<td>rereceive_count</td>
<td>Counter to ensure messages are applied only once at the current database.</td>
</tr>
</tbody>
</table>
**sr_subscriptions view**

Each row lists information about a subscription.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication to which the user ID is subscribed.</td>
</tr>
<tr>
<td>user_name</td>
<td>The user ID that is subscribed to the publication.</td>
</tr>
<tr>
<td>subscribe_by</td>
<td>For publications with a SUBSCRIBE BY expression, this column holds the matching value for this subscription.</td>
</tr>
<tr>
<td>created</td>
<td>The offset in the transaction log at which the subscription was created.</td>
</tr>
<tr>
<td>started</td>
<td>The offset in the transaction log at which the subscription was started.</td>
</tr>
</tbody>
</table>
Stable Queue tables

This section describes the database tables used by SQL Remote to define and manage the stable queue information. The stable queue may be kept in the same database as the SQL Remote database, or in a separate database.

The stable queue is used only by SQL Remote for Adaptive Server Enterprise.

sr_queue_state table

Function

A single row table that stores persistent global information about the state of the stable queue.

Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>SQL Remote version number</td>
</tr>
<tr>
<td>page_id</td>
<td>Transaction log page_id of the last entry scanned.</td>
</tr>
<tr>
<td>row_id</td>
<td>Transaction log row_id of the last entry scanned.</td>
</tr>
<tr>
<td>confirm_offset</td>
<td>The minimum value of the confirmation offsets received from all remote users. This value is used by the Message Agent to decide which transactions can be deleted from the stable queue.</td>
</tr>
<tr>
<td>commit_offset</td>
<td>The transaction log offset of the most recent transaction completed before the oldest incomplete transaction.</td>
</tr>
<tr>
<td>backup_offset</td>
<td>The transaction log offset of the last dump database or dump transaction command. This information is used when the Message Agent is run with the -u command-line switch (replicate only backed up transactions).</td>
</tr>
</tbody>
</table>

sr_queue_state

The most recent incoming message that has been scanned into the stable queue. When a message is applied to the Adaptive Server Enterprise server, it sets the time_received column in sr_remoteuser. When the transaction log is scanned and the transactions from that message are scanned into the stable queue, it sets the time_received column of sr_queue_state.

The purpose of the column is co-ordination between one Message Agent that is scanning the transaction log continuously and another Message Agent that is receiving messages and sending messages in batch mode. When in batch mode, the Message Agent receives messages, waits for those messages to be scanned into the stable queue, and then sends messages. The waiting is done through the database by looking at the time_received column of sr_queue_state.
### sr_transaction table

This table has one row for each transaction in the stable queue.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>The transaction log offset of the commit operation for the transaction. This value uniquely identifies each transaction.</td>
</tr>
<tr>
<td>user_id</td>
<td>The remote user where the transaction originated. This column holds NULL if the transaction did not originate from a remote user. The <code>user_id</code> column is used to ensure that actions are not replicated back to the remote site that entered them.</td>
</tr>
<tr>
<td>data</td>
<td>The transaction itself, in an internal representation.</td>
</tr>
</tbody>
</table>
Stable Queue tables
CHAPTER 17

Command Reference for Adaptive Server Anywhere

About this chapter

This chapter describes the SQL statements used for executing SQL Remote commands, and the system tables, used for storing information about the SQL Remote installation and its state.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER PUBLICATION statement</td>
<td>353</td>
</tr>
<tr>
<td>ALTER REMOTE MESSAGE TYPE statement</td>
<td>354</td>
</tr>
<tr>
<td>CREATE PUBLICATION statement</td>
<td>356</td>
</tr>
<tr>
<td>CREATE REMOTE MESSAGE TYPE statement</td>
<td>358</td>
</tr>
<tr>
<td>CREATE SUBSCRIPTION statement</td>
<td>360</td>
</tr>
<tr>
<td>CREATE TRIGGER statement</td>
<td>361</td>
</tr>
<tr>
<td>DROP PUBLICATION statement</td>
<td>364</td>
</tr>
<tr>
<td>DROP REMOTE MESSAGE TYPE statement</td>
<td>365</td>
</tr>
<tr>
<td>DROP SUBSCRIPTION statement</td>
<td>366</td>
</tr>
<tr>
<td>GRANT CONSOLIDATE statement</td>
<td>367</td>
</tr>
<tr>
<td>GRANT PUBLISH statement</td>
<td>369</td>
</tr>
<tr>
<td>GRANT REMOTE statement</td>
<td>370</td>
</tr>
<tr>
<td>GRANT REMOTE DBA statement</td>
<td>372</td>
</tr>
<tr>
<td>PASSTHROUGH statement</td>
<td>373</td>
</tr>
<tr>
<td>REMOTE RESET statement</td>
<td>374</td>
</tr>
<tr>
<td>REVOKE CONSOLIDATE statement</td>
<td>375</td>
</tr>
<tr>
<td>REVOKE PUBLISH statement</td>
<td>376</td>
</tr>
<tr>
<td>REVOKE REMOTE statement</td>
<td>377</td>
</tr>
<tr>
<td>REVOKE REMOTE DBA statement</td>
<td>378</td>
</tr>
<tr>
<td>START SUBSCRIPTION statement</td>
<td>379</td>
</tr>
</tbody>
</table>
ALTER PUBLICATION statement

STOP SUBSCRIPTION statement 380
SYNCHRONIZE SUBSCRIPTION statement 381
UPDATE statement 382
ALTER PUBLICATION statement

Function
To alter the definition of a SQL Remote publication.

Syntax
```
ALTER PUBLICATION [ owner. ] publication-name
  ADD TABLE article-description
  | MODIFY TABLE article-description
  | ( DELETE | DROP ) TABLE [ owner. ] table-name
  | RENAME publication-name
```

Parameters
- `article-description`: table-name [ ( column-name, ... ) ]
- `WHERE search-condition`
- `SUBSCRIBE BY expression`

Usage
Anywhere. This statement is applicable only to SQL Remote.

Permissions
Must have DBA authority, or be owner of the publication. Requires exclusive access to all tables referred to in the statement.

Side effects
Automatic commit.

See also
"CREATE PUBLICATION statement" on page 356
"DROP PUBLICATION statement" on page 364

Description
The ALTER PUBLICATION statement alters a SQL Remote publication in the database. The contribution to a publication from one table is called an article. Changes can be made to a publication by adding, modifying, or deleting articles, or by renaming the publication. If an article is modified, the entire specification of the modified article must be entered.

Example
The following statement adds the `customer` table to the `pub_contact` publication.

```
ALTER PUBLICATION pub_contact {
  ADD TABLE customer
}
```
ALTER REMOTE MESSAGE TYPE statement

Function
To change the publisher's address for a given message system, for a message type that has been created.

Syntax
ALTER REMOTE MESSAGE TYPE message-system
  ADDRESS address-string

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message-system</td>
<td>One of the message systems supported by SQL Remote. It must be one of the following values:</td>
</tr>
<tr>
<td></td>
<td>♦ FILE</td>
</tr>
<tr>
<td></td>
<td>♦ FTP</td>
</tr>
<tr>
<td></td>
<td>♦ MAPI</td>
</tr>
<tr>
<td></td>
<td>♦ SMTP</td>
</tr>
<tr>
<td></td>
<td>♦ VIM</td>
</tr>
<tr>
<td>address-string</td>
<td>A string containing a valid address for the specified message system.</td>
</tr>
</tbody>
</table>

Permissions
Must have DBA authority.

Side effects
Automatic commit.

See also
"CREATE REMOTE MESSAGE TYPE statement" on page 358

Description
The statement changes the publisher's address for a given message type.

The Message Agent sends outgoing messages from a database by one of the supported message links. The extraction utility uses this address when executing the GRANT CONSOLIDATE statement in the remote database.

The address is the publisher's address under the specified message system. If it is an e-mail system, the address string must be a valid e-mail address. If it is a file-sharing system, the address string is a subdirectory of the directory specified by the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.

For the FILE link, the ALTER REMOTE MESSAGE TYPE statement also causes the Message Agent to look for incoming messages in the address given for each message type.

Example
The following statement changes the publisher's address for the FILE message link to new_addr.

CREATE REMOTE MESSAGE TYPE file
ADDRESS 'new_addr'
CREATE PUBLICATION statement

Function
To create a publication for replication with SQL Remote.

Syntax
CREATE PUBLICATION [ owner.]publication-name
  ... ( TABLE article-description,... )

Parameters
article-description:
  table-name [ ( column-name, ... ) ]
  ... [ WHERE search-condition ]
  ... [ SUBSCRIBE BY expression ]

Permissions
Must have DBA authority. Requires exclusive access to all tables referred to in the statement.

Side effects
Automatic commit.

See also
"ALTER PUBLICATION statement" on page 353
"DROP PUBLICATION statement" on page 364

Description
The CREATE PUBLICATION statement creates a publication in the database. A publication can be created for another user by specifying an owner name.

In SQL Remote, publishing is a two-way operation, as data can be entered at both consolidated and remote databases. In a SQL Remote installation, any consolidated database and all remote databases must have the same publication defined. Running the extraction utility from a consolidated database automatically executes the correct CREATE PUBLICATION statement in the remote database.

Article Publications are built from articles. Each article is a table or part of a table. An article may be a vertical partition of a table (a subset of the table's columns), a horizontal partition (a subset of the table's rows) or a vertical and horizontal partition.

SUBSCRIBE BY clause One way of defining a subset of rows of a table to be included in an article is to use a SUBSCRIBE BY clause. This clause allows many different subscribers to receive different rows from a table in a single publication definition.

WHERE clause The WHERE clause is a way of defining the subset of rows of a table to be included in an article. It is useful if the same subset if to be received by all subscribers to the publication.

You can combine WHERE and SUBSCRIBE BY clauses in an article definition.

Example
♦ The following statement creates a simple publication:

CREATE PUBLICATION pub_contact (}
TABLE contact
}


CREATE REMOTE MESSAGE TYPE statement

Function
To identify a message-link and return address for outgoing messages from a database.

Syntax
CREATE REMOTE MESSAGE TYPE message-system
...ADDRESS address-string

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message-system</td>
<td>One of the message systems supported by SQL Remote. It must be one of the following values:</td>
</tr>
<tr>
<td></td>
<td>♦ FILE</td>
</tr>
<tr>
<td></td>
<td>♦ FTP</td>
</tr>
<tr>
<td></td>
<td>♦ MAPI</td>
</tr>
<tr>
<td></td>
<td>♦ SMTP</td>
</tr>
<tr>
<td></td>
<td>♦ VIM</td>
</tr>
<tr>
<td>address-string</td>
<td>A string containing a valid address for the specified message system.</td>
</tr>
</tbody>
</table>

Permissions
Must have DBA authority.

Side effects
Automatic commit.

See also
"GRANT PUBLISH statement" on page 369
"GRANT REMOTE statement" on page 370
"GRANT CONSOLIDATE statement" on page 367
"Using message types" on page 228

Description
The Message Agent sends outgoing messages from a database using one of the supported message links. Return messages for users employing the specified link are sent to the specified address as long as the remote database is created by the extraction utility. The Message Agent starts links only if it has remote users for those links.

The address is the publisher's address under the specified message system. If it is an e-mail system, the address string must be a valid e-mail address. If it is a file-sharing system, the address string is a subdirectory of the directory set in the SQLREMOTE environment variable, or of the current directory if that is not set. You can override this setting on the GRANT CONSOLIDATE statement at the remote database.

For the FILE link, the CREATE REMOTE MESSAGE TYPE statement also causes the Message Agent to look for incoming messages in the address given for each message type.
The initialization utility creates message types automatically, without an address. Unlike other CREATE statements, the CREATE REMOTE MESSAGE TYPE statement does not give an error if the type exists; instead it alters the type.

**Example**

- When remote databases are extracted using the extraction utility, the following statement sets all recipients of FILE messages to send messages back to the company subdirectory of the SQLREMOTE environment variable.

  The statement also instructs DBREMOTE to look in the *company* subdirectory for incoming messages.

  ```sql
  CREATE REMOTE MESSAGE TYPE file
  ADDRESS 'company'
  ```
CREATE SUBSCRIPTION statement

Function
To create a subscription for a user to a publication.

Syntax
CREATE SUBSCRIPTION
    ... TO publication-name [ ( subscription-value ) ]
    ... FOR subscriber-id

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication-name</td>
<td>The name of the publication to which the user is being subscribed. This may include the owner of the publication.</td>
</tr>
<tr>
<td>subscription-value</td>
<td>A string that is compared to the subscription expression of the publication. The subscriber receives all rows for which the subscription expression matches the subscription value.</td>
</tr>
<tr>
<td>subscriber-id</td>
<td>The user ID of the subscriber to the publication. This user must have been granted REMOTE permissions.</td>
</tr>
</tbody>
</table>

Permissions
Must have DBA authority.

Side effects
Automatic commit.

See also
"DROP SUBSCRIPTION statement" on page 366
"GRANT REMOTE statement" on page 370
"SYNCHRONIZE SUBSCRIPTION statement" on page 381
"START SUBSCRIPTION statement" on page 379

Description
In a SQL Remote installation, data is organized into publications for replication. In order to receive SQL Remote messages, a subscription must be created for a user ID with REMOTE permissions.

If a string is supplied in the subscription, it is matched against each SUBSCRIBE BY expression in the publication. The subscriber receives all rows for which the value of the expression is equal to the supplied string.

In SQL Remote, publications and subscriptions are two-way relationships. If you create a subscription for a remote user to a publication on a consolidated database, you should also create a subscription for the consolidated database on the remote database. The extraction utility carries this out automatically.

Example
♦ The following statement creates a subscription for the user p_chin to the publication pub_sales. The subscriber receives all rows for which the subscription expression has a value of Eastern.

```sql
CREATE SUBSCRIPTION
    TO pub_sales ( 'Eastern' )
    FOR p_chin
```
CREATE TRIGGER statement

Function
To create a new trigger in the database. One form of trigger is designed specifically for use by SQL Remote.

Syntax
CREATE TRIGGER trigger-name trigger-time trigger-event
  [, trigger-event...]  
  ... [ ORDER integer ] ON table-name
  ... [ REFERENCING [ OLD AS old-name ]
    [ NEW AS new-name ]
    [ REMOTE AS remote-name ]]
  ... [ FOR EACH ( ROW | STATEMENT ) ]
  ... [ WHEN ( search-condition ) ]
  ... [ IF UPDATE ( column-name ) THEN
    ... [ { AND | OR } UPDATE ( column-name ) ] ... ]
    compound-statement
  ... [ ELSEIF UPDATE ( column-name ) THEN
    ... [ { AND | OR } UPDATE ( column-name ) ] ... ]
    compound-statement
  ... END IF ]]

Parameters
trigger-time:
  BEFORE | AFTER | RESOLVE

trigger-event:
  DELETE | INSERT | UPDATE | UPDATE OF column-list

Usage
Anywhere.

Permissions
Must have RESOURCE authority and have ALTER permissions on the table, or must have DBA authority. CREATE TRIGGER puts a table lock on the table and thus requires exclusive use of the table.

Side effects
Automatic commit.

See also
"UPDATE statement" on page 382

Description
The CREATE TRIGGER statement creates a trigger associated with a table in the database and stores the trigger in the database.

Triggers can be triggered by one or more of the following events:

♦ **INSERT** Invoked whenever a new row is inserted into the table associated with the trigger.

♦ **DELETE** Invoked whenever a row of the associated table is deleted.

♦ **UPDATE** Invoked whenever a row of the associated table is updated.

♦ **UPDATE OF column-list** Invoked whenever a row of the associated table is updated and a column in the column-list has been modified.
### CREATE TRIGGER statement

**BEFORE UPDATE** triggers fire any time an update occurs on a row, regardless of whether or not the new value differs from the old value. **AFTER UPDATE** triggers will fire only if the new value is different from the old value.

**Row and statement-level triggers**

The trigger is declared as either a row-level trigger, in which case it executes before or after each row is modified, or as a statement-level trigger, in which case it executes after the entire triggering statement is completed.

Row-level triggers can be defined to execute **BEFORE** or **AFTER** the insert, update, or delete. Statement-level triggers execute **AFTER** the statement. The **RESOLVE** trigger time is for use with SQL Remote; it fires before row-level UPDATE or UPDATE OF column-lists only.

To declare a trigger as a row-level trigger, use the **FOR EACH ROW** clause. To declare a trigger as a statement-level trigger, you can either use a **FOR EACH STATEMENT** clause or omit the FOR EACH clause. For clarity, it is recommended that you enter the **FOR EACH STATEMENT** clause if declaring a statement-level trigger.

**Order of firing**

Triggers of the same type (insert, update, or delete) that fire at the same time (before, after, or resolve) can use the ORDER clause to determine the order that the triggers are fired.

**Referencing deleted and inserted values**

The **REFERENCING OLD** and **REFERENCING NEW** clauses allow you to refer to the deleted and inserted rows. For the purposes of this clause, an **UPDATE** is treated as a delete followed by an insert.

The **REFERENCING REMOTE** clause is for use with SQL Remote. It allows you to refer to the values in the **VERIFY** clause of an **UPDATE** statement. It should be used only with **RESOLVE UPDATE** or **RESOLVE UPDATE OF** column-list triggers.

The meaning of **REFERENCING OLD** and **REFERENCING NEW** differs, depending on whether the trigger is a row-level or a statement-level trigger. For row-level triggers, the **REFERENCING OLD** clause allows you to refer to the values in a row prior to an update or delete, and the **REFERENCING NEW** clause allows you to refer to the inserted or updated values. The **OLD** and **NEW** rows can be referenced in **BEFORE** and **AFTER** triggers. The **REFERENCING NEW** clause allows you to modify the new row in a **BEFORE** trigger before the insert or update operation takes place.

For statement-level triggers, the **REFERENCING OLD** and **REFERENCING NEW** clauses refer to declared temporary tables holding the old and new values of the rows. The default names for these tables are **deleted** and **inserted**.

The **WHEN** clause causes the trigger to fire only for rows where the search-condition evaluates to true.
<table>
<thead>
<tr>
<th>Updating values with the same value</th>
<th>BEFORE UPDATE triggers fire any time an UPDATE occurs on a row, whether or not the new value differs from the old value. AFTER UPDATE triggers fire only if the new value is different from the old value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>♦ When a new department head is appointed, update the <code>manager_id</code> column for employees in that department.</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
<tr>
<td></td>
<td>CREATE TRIGGER tr_manager BEFORE UPDATE OF dept_head_id ON department</td>
</tr>
<tr>
<td></td>
<td>REFERENCING OLD AS old_dept</td>
</tr>
<tr>
<td></td>
<td>NEW AS new_dept</td>
</tr>
<tr>
<td></td>
<td>FOR EACH ROW</td>
</tr>
<tr>
<td></td>
<td>BEGIN</td>
</tr>
<tr>
<td></td>
<td>UPDATE employee</td>
</tr>
<tr>
<td></td>
<td>SET employee.manager_id=new_dept.dept_head_id</td>
</tr>
<tr>
<td></td>
<td>WHERE employee.dept_id=old_dept.dept_id</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
<tr>
<td></td>
<td>```</td>
</tr>
</tbody>
</table>
DROP PUBLICATION statement

Function
To drop a SQL Remote publication.

Syntax
DROP PUBLICATION [ owner.]publication-name

Usage
Anywhere. This statement is applicable only to SQL Remote.

Permissions
Must have DBA authority.

Side effects
Automatic commit. All subscriptions to the publication are dropped.

See also
"CREATE PUBLICATION statement" on page 356

Description
The DROP PUBLICATION statement drops an existing publication from the database.

Publication definitions are held at both consolidated and remote databases in a SQL Remote installation.

Example
DROP PUBLICATION pub_contact
## DROP REMOTE MESSAGE TYPE statement

### Function
To delete a message type definition from a database.

### Syntax
```
DROP REMOTE MESSAGE TYPE message-system
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `message-system` | One of the message systems supported by SQL Remote. It must be one of the following values:     
| ♦ FILE       |             |
| ♦ FTP        |             |
| ♦ MAPI       |             |
| ♦ SMTP       |             |
| ♦ VIM        |             |

### Permissions
Must have DBA authority. To be able to drop the type, there must be no user granted REMOTE or CONSOLIDATE permissions with this type.

### Side effects
Automatic commit.

### See also
- "CREATE REMOTE MESSAGE TYPE statement" on page 358
- "ALTER REMOTE MESSAGE TYPE statement" on page 354
- "Using message types" on page 228.

### Description
The statement removes a message type from a database.

### Example
The following statement drops the FILE message type from a database.

```
DROP REMOTE MESSAGE TYPE file
```
DROP SUBSCRIPTION statement

<table>
<thead>
<tr>
<th>Function</th>
<th>To drop a subscription for a user from a publication.</th>
</tr>
</thead>
</table>
| Syntax   | DROP SUBSCRIPTION TO publication-name [( subscription-value )] ...
|          | FOR subscriber-id,... |
| Parameters | Parameter | Description                                      |
|           | publication-name | The name of the publication to which the user is being subscribed. This may include the owner of the publication. |
|           | subscription-value | A string that is compared to the subscription expression of the publication. This value is required because a user may have more than one subscription to a publication. |
|           | subscriber-id | The user ID of the subscriber to the publication. |
| Permissions | Must have DBA authority. |
| Side effects | Automatic commit. |
| See also | "CREATE SUBSCRIPTION statement" on page 360 |
| Description | Drops a SQL Remote subscription for a user ID to a publication in the current database. The user ID will no longer receive updates when data in the publication is changed. |
|            | In SQL Remote, publications and subscriptions are two-way relationships. If you drop a subscription for a remote user to a publication on a consolidated database, you should also drop the subscription for the consolidated database on the remote database to prevent updates on the remote database being sent to the consolidated database. |
| Example   | The following statement drops a subscription for the user ID SamS to the publication pub_contact. |
|           | DROP SUBSCRIPTION TO pub_contact |
|           | FOR SamS |
GRANT CONSOLIDATE statement

Function
To identify the database immediately above the current database in a SQL Remote hierarchy, who will receive messages from the current database.

Syntax
```
GRANT CONSOLIDATE
    ... TO userid, ...
    ... TYPE message-system, ...
    ... ADDRESS address-string, ...
    ... [ SEND { EVERY | AT }'hh:mm' ]
```

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userid</td>
<td>The user ID for the user to be granted the permission</td>
</tr>
<tr>
<td>message-system</td>
<td>One of the message systems supported by SQL Remote. It must be one of the following values:</td>
</tr>
<tr>
<td></td>
<td>♦ FILE</td>
</tr>
<tr>
<td></td>
<td>♦ FTP</td>
</tr>
<tr>
<td></td>
<td>♦ MAPI</td>
</tr>
<tr>
<td></td>
<td>♦ SMTP</td>
</tr>
<tr>
<td></td>
<td>♦ VIM</td>
</tr>
<tr>
<td>address-string</td>
<td>A string containing a valid address for the specified message system.</td>
</tr>
</tbody>
</table>

Permissions
Must have DBA authority.

Side effects
Automatic commit.

See also
"GRANT REMOTE statement" on page 370
"REVOKE CONSOLIDATE statement" on page 375
"GRANT PUBLISH statement" on page 369

Description
In a SQL Remote installation, the database immediately above the current database in a SQL Remote hierarchy must be granted CONSOLIDATE permissions. GRANT CONSOLIDATE is issued at a remote database to identify its consolidated database. Each database can have only one user ID with CONSOLIDATE permissions: you cannot have a database that is a remote database for more than one consolidated database.

The consolidated user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The address-name must be a valid address for the message-system, enclosed in single quotes.
For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.

The GRANT CONSOLIDATE statement is required for the consolidated database to receive messages, but does not by itself subscribe the consolidated database to any data. To subscribe to data, a subscription must be created for the consolidated user ID to one of the publications in the current database. Running the database extraction utility at a consolidated database creates a remote database with the proper GRANT CONSOLIDATE statement already issued.

The optional SEND EVERY and SEND AT clauses specify a frequency at which messages are sent. The string contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If a user has been granted remote permissions without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. In order to run the Message Agent continuously, you must ensure that every user with REMOTE permission has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many remote databases, the Message Agent will be run periodically, and that the consolidated database will have no SEND clause specified.

**Example**

```
GRANT CONSOLIDATE TO con_db
TYPE mapi
ADDRESS 'Consolidated Database'
```
**GRANT PUBLISH statement**

**Function**
To identify the publisher of the current database.

**Syntax**

GRANT PUBLISH TO userid

**Permissions**
Must have DBA authority.

**Side effects**
Automatic commit.

**See also**
"GRANT REMOTE statement" on page 370
"GRANT CONSOLIDATE statement" on page 367
"REVOKE PUBLISH statement" on page 376
"CREATE PUBLICATION statement" on page 356
"CREATE SUBSCRIPTION statement" on page 360

**Description**
Each database in a SQL Remote installation is identified in outgoing messages by a user ID, called the **publisher**. The GRANT PUBLISH statement identifies the publisher user ID associated with these outgoing messages.

Only one user ID can have PUBLISH authority. The user ID with PUBLISH authority is identified by the special constant CURRENT PUBLISHER. The following query identifies the current publisher:

```
SELECT CURRENT PUBLISHER
```

If there is no publisher, the special constant is NULL.

The current publisher special constant can be used as a default setting for columns. It is often useful to have a CURRENT PUBLISHER column as part of the primary key for replicating tables, as this helps prevent primary key conflicts due to updates at more than one site.

In order to change the publisher, you must first drop the current publisher using the REVOKE PUBLISH statement, and then create a new publisher using the GRANT PUBLISH statement.

**Example**

```
GRANT PUBLISH TO publisher_ID
```
GRANT REMOTE statement

Function
To identify a database immediately below the current database in a SQL Remote hierarchy, who will receive messages from the current database. These are called remote users.

Syntax

\[
\text{GRANT REMOTE TO } \textit{userid}, \ldots \\
\ldots \text{ TYPE } \textit{message-system}, \ldots \\
\ldots \text{ ADDRESS } \textit{address-string}, \ldots \\
\ldots \text{ [ SEND } \{ \textit{EVERY | AT} \textit{send-time} \} \\
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{userid}</td>
<td>The user ID for the user to be granted the permission</td>
</tr>
<tr>
<td>\textit{message-system}</td>
<td>One of the message systems supported by SQL Remote. It must be one of the following values:</td>
</tr>
<tr>
<td></td>
<td>♦ FILE</td>
</tr>
<tr>
<td></td>
<td>♦ FTP</td>
</tr>
<tr>
<td></td>
<td>♦ MAPI</td>
</tr>
<tr>
<td></td>
<td>♦ SMTP</td>
</tr>
<tr>
<td></td>
<td>♦ VIM</td>
</tr>
<tr>
<td>\textit{address-string}</td>
<td>A string containing a valid address for the specified message system.</td>
</tr>
<tr>
<td>\textit{send-time}</td>
<td>A string containing a time specification in the form \textit{hh:mm}.</td>
</tr>
</tbody>
</table>

Permissions
Must have DBA authority.

Side effects
Automatic commit.

See also
"GRANT CONSOLIDATE statement" on page 367
"REVOKE REMOTE statement" on page 377
"GRANT PUBLISH statement" on page 369
"Granting and revoking REMOTE and CONSOLIDATE permissions" on page 221,

Description
In a SQL Remote installation, each database receiving messages from the current database must be granted REMOTE permissions.

The single exception is the database immediately above the current database in a SQL Remote hierarchy, which must be granted CONSOLIDATE permissions.
The remote user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The address-name must be a valid address for the message-system, enclosed in single quotes.

For the FILE message type, the address is a subdirectory of the directory pointed to by the SQLREMOTE environment variable.

The GRANT REMOTE statement is required for the remote database to receive messages, but does not by itself subscribe the remote user to any data. To subscribe to data, a subscription must be created for the user ID to one of the publications in the current database, using the database extraction utility or the CREATE SUBSCRIPTION statement.

The optional SEND EVERY and SEND AT clauses specify a frequency at which messages are sent. The string contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If a user has been granted remote permissions without a SEND EVERY or SEND AT clause, the Message Agent processes messages, and then stops. In order to run the Message Agent continuously, you must ensure that every user with REMOTE permission has either a SEND AT or SEND EVERY frequency specified.

It is anticipated that at many consolidated databases, the Message Agent will be run continuously, so that all remote databases would have a SEND clause specified. A typical setup may involve sending messages to laptop users daily (SEND AT) and to remote servers every hour or two (SEND EVERY). You should use as few different times as possible, for efficiency.

Example

- The following statement grants remote permissions to user SamS, using a MAPI e-mail system, sending messages to the address Singer, Samuel once every two hours:

```
GRANT REMOTE TO SamS
  TYPE mapi
  ADDRESS 'Singer, Samuel'
  SEND EVERY '02:00'
```
**GRANT REMOTE DBA statement**

**Function**
To provide DBA privileges to a user ID, but only when connected from the Message Agent.

**Syntax 1**

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

**Permissions**
Must have DBA authority.

**Side effects**
Automatic commit.

**See also**
"The Message Agent and replication security" on page 257
"REVOKE REMOTE DBA statement" on page 378

**Description**
REMOTE DBA authority enables the Message Agent to have full access to the database in order to make any changes contained in the messages, while avoiding security problems associated with distributing DBA user IDs passwords.

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. 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 when connected from the Message Agent.
PASSTHROUGH statement

Function
To start or stop passthrough mode for SQL Remote administration. Forms 1 and 2 start passthrough mode, while form 3 stops passthrough mode.

Syntax 1
PASSTHROUGH [ ONLY ] FOR userid,...

Syntax 2
PASSTHROUGH [ ONLY ] FOR SUBSCRIPTION
... TO { ( owner ) }.publication-name [ ( constant ) ]

Syntax 3
PASSTHROUGH STOP

Permissions
Must have DBA authority.

Side effects
None.

Description
In passthrough mode, any SQL statements are executed by the database server, and are also placed into the transaction log to be sent in messages to subscribers. If the ONLY keyword is used to start passthrough mode, the statements are not executed at the server; they are sent to recipients only. The recipients of the passthrough SQL statements are either a list of user IDs (form 1) or all subscribers to a given publication. Passthrough mode may be used to apply changes to a remote database from the consolidated database or send statements from a remote database to the consolidated database.

Syntax 2 sends statements to remote databases whose subscriptions are started, and does not send statements to remote databases whose subscriptions are created and not started.

Example
PASSTHROUGH FOR rem_db ;
...
( SQL statements to be executed at the remote database )
...
PASSTHROUGH STOP ;

373
## REMOTE RESET statement

**Function**
For use in custom database extraction procedures. It starts all subscriptions for a remote user in a single transaction.

**Syntax**
REMOTE RESET *userid*

**Permissions**
Must have DBA authority.

**Side effects**
No automatic commit is done by this statement.

**See also**
"START SUBSCRIPTION statement" on page 379

**Description**
This command starts all subscriptions for a remote user in a single transaction. It sets the log_sent and confirm_sent values in SYSREMOTEUSER table to the current position in the transaction log. It also sets the created and started values in SYSSUBSCRIPTION to the current position in the transaction log for all subscriptions for this remote user. The statement does not do a commit. You must do an explicit commit after this call.

In order to write an extraction process that is safe on a live database, the data must be extracted at isolation level 3 in the same transaction as the subscriptions are started.

This statement is an alternative to start subscription. START SUBSCRIPTION has an implicit commit as a side effect, so that if a remote user has several subscriptions, it is impossible to start them all in one transaction using START SUBSCRIPTION.

**Example**
- The following statement resets the subscriptions for remote user SamS:

  ```sql
  REMOTE RESET 'SamS'
  ```

374
## REVOKE CONSOLIDATE statement

<table>
<thead>
<tr>
<th>Function</th>
<th>To stop a consolidated database from receiving SQL Remote messages from this database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>REVOKE CONSOLIDATE FROM userid,...</code></td>
</tr>
<tr>
<td>Permissions</td>
<td>Must have DBA authority.</td>
</tr>
<tr>
<td>Side effects</td>
<td>Automatic commit. Drops all subscriptions for the user.</td>
</tr>
<tr>
<td>See also</td>
<td>&quot;GRANT CONSOLIDATE statement&quot; on page 367</td>
</tr>
<tr>
<td>Description</td>
<td>CONSOLIDATE permissions must be granted at a remote database for the user ID representing the consolidated database. The REVOKE CONSOLIDATE statement removes the consolidated database user ID from the list of users receiving messages from the current database.</td>
</tr>
<tr>
<td>Example</td>
<td>♦ The following statement revokes consolidated status from the user ID <code>condb</code>:</td>
</tr>
<tr>
<td></td>
<td><code>REVOKE CONSOLIDATE FROM condb</code></td>
</tr>
</tbody>
</table>
REVOKE PUBLISH statement

Function
To terminate the identification of the named user ID as the CURRENT publisher.

Syntax
REVOKE PUBLISH FROM \textit{userid}

Permissions
Must have DBA authority.

Side effects
Automatic commit.

See also
"GRANT PUBLISH statement" on page 369
"REVOKE REMOTE statement" on page 377
"CREATE PUBLICATION statement" on page 356
"CREATE SUBSCRIPTION statement" on page 360

Description
Each database in a SQL Remote installation is identified in outgoing messages by a \textit{publisher} user ID. The current publisher user ID can be found using the CURRENT PUBLISHER special constant. The following query identifies the current publisher:

\begin{verbatim}
SELECT CURRENT PUBLISHER
\end{verbatim}

The REVOKE PUBLISH statement ends the identification of the named user ID as the publisher.

You should not REVOKE PUBLISH from a database while the database has active SQL Remote publications or subscriptions.

Issuing a REVOKE PUBLISH statement at a database has several consequences for a SQL Remote installation:

\begin{itemize}
  \item You will not be able to insert data into any tables with a CURRENT PUBLISHER column as part of the primary key. Any outgoing messages will not be identified with a publisher user ID, and so will not be accepted by recipient databases.
\end{itemize}

If you change the publisher user ID at any consolidated or remote database in a SQL Remote installation, you must ensure that the new publisher user ID is granted REMOTE permissions on all databases receiving messages from the database. This will generally require all subscriptions to be dropped and recreated.

Example
REVOKE PUBLISH FROM publisher\_ID
REVOKE REMOTE statement

**Function**
To stop a user from being able to receive SQL Remote messages from this database.

**Syntax**
REVOKE REMOTE FROM *userid,...*

**Permissions**
Must have DBA authority.

**Side effects**
Automatic commit. Drops all subscriptions for the user.

**Description**
REMOTE permissions are required for a user ID to receive messages in a SQL Remote replication installation. The REVOKE REMOTE statement removes a user ID from the list of users receiving messages from the current database.

**Example**
REVOKE REMOTE FROM SamS
## REVOKE REMOTE DBA statement

**Function**
To provide DBA privileges to a user ID, but only when connected from the Message Agent.

**Syntax 1**
```
REVOKE REMOTE DBA
   FROM userid,...
```

**Permissions**
Must have DBA authority.

**Side effects**
Automatic commit.

**See also**
"The Message Agent and replication security" on page 257
"GRANT REMOTE DBA statement" on page 372

**Description**
REMOTE DBA authority enables the Message Agent to have full access to the database in order to make any changes contained in the messages, while avoiding security problems associated with distributing DBA user IDs passwords.

This statement revokes REMOTE DBA authority from a user ID.
START SUBSCRIPTION statement

Function
To start a subscription for a user to a publication.

Syntax
START SUBSCRIPTION
   ... TO publication-name [ ( subscription-value) ]
   ... FOR subscriber-id,...

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication-name</td>
<td>The name of the publication to which the user is being subscribed. This may include the owner of the publication.</td>
</tr>
<tr>
<td>subscription-value</td>
<td>A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.</td>
</tr>
<tr>
<td>subscriber-id</td>
<td>The user ID of the subscriber to the publication. This user must have a subscription to the publication.</td>
</tr>
</tbody>
</table>

Permissions
Must have DBA authority.

Side effects
Automatic commit.

See also
"CREATE SUBSCRIPTION statement" on page 360
"REMOTE RESET statement" on page 374
"SYNCHRONIZE SUBSCRIPTION statement" on page 381

Description
A SQL Remote subscription is said to be **started** when publication updates are being sent from the consolidated database to the remote database.

The START SUBSCRIPTION statement is one of a set of statements that manage subscriptions. The CREATE SUBSCRIPTION statement defines the data that the subscriber is to receive. The SYNCHRONIZE SUBSCRIPTION statement ensures that the consolidated and remote databases are consistent with each other. The START SUBSCRIPTION statement is required to start messages being sent to the subscriber.

Data at each end of the subscription must be consistent before a subscription is started. It is recommended that you use the database extraction utility to manage the creation, synchronization, and starting of subscriptions. If you use the database extraction utility, you do not need to execute an explicit START SUBSCRIPTION statement. Also, the Message Agent starts subscriptions once they are synchronized.

Example
The following statement starts the subscription of user **SamS** to the **pub_contact** publication.

```
START SUBSCRIPTION TO pub_contact
   FOR SamS
```
STOP SUBSCRIPTION statement

Function
To stop a subscription for a user to a publication.

Syntax
```
STOP SUBSCRIPTION
... TO publication-name [ ( subscription-value ) ]
... FOR subscriber-id,...
```

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication-name</td>
<td>The name of the publication to which the user is being subscribed. This may include the owner of the publication.</td>
</tr>
<tr>
<td>subscription-value</td>
<td>A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.</td>
</tr>
<tr>
<td>subscriber-id</td>
<td>The user ID of the subscriber to the publication. This user must have a subscription to the publication.</td>
</tr>
</tbody>
</table>

Permissions
Must have DBA authority.

Side effects
Automatic commit.

See also
"CREATE SUBSCRIPTION statement" on page 360
"SYNCHRONIZE SUBSCRIPTION statement" on page 381

Description
A SQL Remote subscription is said to be started when publication updates are being sent from the consolidated database to the remote database.

The STOP SUBSCRIPTION statement prevents any further messages being sent to the subscriber. The START SUBSCRIPTION statement is required to restart messages being sent to the subscriber. However, you should ensure that the subscription is properly synchronized before restarting; that no messages have been missed.

Example
The following statement starts the subscription of user SamS to the pub_contact publication.

```
STOP SUBSCRIPTION TO pub_contact
FOR SamS
```
SYNCHRONIZE SUBSCRIPTION statement

Function
To synchronize a subscription for a user to a publication.

Syntax
SYNCHRONIZE SUBSCRIPTION
  
  ... TO publication-name [ ( subscription-value ) ]
  
  ... FOR remote-user,...

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication-name</td>
<td>The name of the publication to which the user is being subscribed. This may include the owner of the publication.</td>
</tr>
<tr>
<td>subscription-value</td>
<td>A string that is compared to the subscription expression of the publication. The value is required here because each subscriber may have more than one subscription to a publication.</td>
</tr>
<tr>
<td>remote-user</td>
<td>The user ID of the subscriber to the publication. This user must have a subscription to the publication.</td>
</tr>
</tbody>
</table>

Permissions
Must have DBA authority.

Side effects
Automatic commit.

See also
"CREATE SUBSCRIPTION statement" on page 360
"START SUBSCRIPTION statement" on page 379

Description
A SQL Remote subscription is said to be **synchronized** when the data in the remote database is consistent with that in the consolidated database, so that publication updates sent from the consolidated database to the remote database will not result in conflicts and errors.

To synchronize a subscription, a copy of the data in the publication at the consolidated database is sent to the remote database. The SYNCHRONIZE SUBSCRIPTION statement does this through the message system. It is recommended that where possible you use the database extraction utility instead to synchronize subscriptions without using a message system.

Example
The following statement synchronizes the subscription of user SamS to the pub_contact publication.

SYNCHRONIZE SUBSCRIPTION TO pub_contact
FOR SamS
UPDATE statement

Function To modify data in the database.

Syntax 1

```
UPDATE table-list
...  SET column-name = expression, ...
...  [ FROM table-list ]
...  [ WHERE search-condition ]
...  [ ORDER BY expression [ ASC | DESC ], ... ]
```

Syntax 2

```
UPDATE table-list
...  SET column-name = expression, ...
...  [ VERIFY ( column-name, ... ) VALUES ( expression, ... ) ]
...  [ WHERE search-condition ]
...  [ ORDER BY expression [ ASC | DESC ], ... ]
```

Syntax 3

```
UPDATE table
...{ PUBLICATION publication
...{ SUBSCRIBE BY expression | OLD SUBSCRIBE BY expression
NEW SUBSCRIBE BY expression
}
... WHERE search-condition
```

expression:  value | subquery

Usage Syntax 1 can be used anywhere.
Syntax 2 and Syntax 3 are applicable only to SQL Remote.
Syntax 3 with no OLD and NEW SUBSCRIBE BY expressions must be used in a BEFORE trigger.
Syntax 3 with OLD and NEW SUBSCRIBE BY expressions can be used anywhere.

Permissions Must have UPDATE permission for the columns being modified.

Side effects None.

See also "CREATE TRIGGER statement" on page 361

Description The UPDATE statement is used to modify rows of one or more tables. Each named column is set to the value of the expression on the right hand side of the equal sign. There are no restrictions on the expression. Even column-name can be used in the expression—the old value will be used.

If no WHERE clause is specified, every row will be updated. If a WHERE clause is specified, then only those rows which satisfy the search condition will be updated.
Normally, the order that rows are updated doesn't matter. However, in conjunction with the NUMBER(*) function, an ordering can be useful to get increasing numbers added to the rows in some specified order. Also, if you wish to do something like add 1 to the primary key values of a table, it is necessary to do this in descending order by primary key, so that you do not get duplicate primary keys during the operation.

Views can be updated provided the SELECT statement defining the view does not contain a GROUP BY clause, an aggregate function, or involve a UNION operation.

Character strings inserted into tables are always stored in the case they are entered, regardless of whether the database is case sensitive or not. Thus a character data type column updated with a string \texttt{Value} is always held in the database with an upper-case \texttt{V} and the remainder of the letters lower case. SELECT statements return the string as \texttt{Value}. If the database is not case-sensitive, however, all comparisons make \texttt{Value} the same as \texttt{value}, \texttt{VALUE}, and so on. Further, if a single-column primary key already contains an entry \texttt{Value}, an INSERT of \texttt{value} is rejected, as it would make the primary key not unique.

The optional FROM clause allows tables to be updated based on joins. If the FROM clause is present, the WHERE clause qualifies the rows of the FROM clause. Data is updated only in the table list immediately following the UPDATE keyword.

If a FROM clause is used, it is important to qualify the table name that is being updated the same way in both parts of the statement. If a correlation name is used in one place, the same correlation name must be used in the other. Otherwise, an error is generated.

Syntax 2 is intended for use with SQL Remote only, in single-row updates executed by the Message Agent. The VERIFY clause contains a set of values that are expected to be present in the row being updated. If the values do not match, any RESOLVE UPDATE triggers are fired before the UPDATE proceeds. The UPDATE does not fail if the VERIFY clause fails to match.

Syntax 3 is intended for use with SQL Remote only. If no OLD and NEW expressions are used, it must be used inside a BEFORE trigger so that it has access to the relevant values. The purpose is to provide a full list of subscribe by values any time the list changes. It is placed in SQL Remote triggers so that the database server can compute the current list of SUBSCRIBE BY values. Both lists are placed in the transaction log.

The Message Agent uses the two lists to make sure that the row moves to any remote database that did not have the row and now needs it. The Message Agent also removes the row from any remote database that has the row and no longer needs it. A remote database that has the row and still needs it is not be affected by the UPDATE statement.
**UPDATE statement**

**Performance tip**
Syntax 3 of the UPDATE statement allows the old SUBSCRIBE BY list and the new SUBSCRIBE BY list to be explicitly specified, which can make SQL Remote triggers more efficient. In the absence of these lists, the database server computes the old SUBSCRIBE BY list from the publication definition. Since the new SUBSCRIBE BY list is commonly only slightly different from the old SUBSCRIBE BY list, the work to compute the old list may be done twice. By specifying both the old and new lists, this extra work can be avoided.

The OLD and NEW SUBSCRIBE BY syntax is especially useful when many tables are being updated in the same trigger with the same subscribe by expressions. This can dramatically increase performance.

The SUBSCRIBE BY expression is either a value or a subquery.

**Using UPDATE to maintain subscriptions**
Syntax 3 of the UPDATE statement is used to implement a specific SQL Remote feature, and is to be used inside a BEFORE trigger.

For publications created using a subquery in a subscription expression, you must write a trigger containing syntax 3 of the UPDATE statement in order to ensure that the rows are kept in their proper subscriptions.

☞ For a full description of this feature, see "Territory realignment in the Contact example" on page 128.

**Examples**

- Transfer employee Philip Chin (employee 129) from the sales department to the marketing department.

```sql
UPDATE employee
SET dept_id = 400
WHERE emp_id = 129 ;
```
CHAPTER 18

Command Reference for Adaptive Server Enterprise

About this chapter

This chapter describes the SQL Remote stored procedures, used for executing SQL Remote commands.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_add_article procedure</td>
<td>387</td>
</tr>
<tr>
<td>sp_add_article_col procedure</td>
<td>388</td>
</tr>
<tr>
<td>sp_add_remote_table procedure</td>
<td>389</td>
</tr>
<tr>
<td>sp_create_publication procedure</td>
<td>391</td>
</tr>
<tr>
<td>sp_drop_publication procedure</td>
<td>392</td>
</tr>
<tr>
<td>sp_drop_remote_type procedure</td>
<td>393</td>
</tr>
<tr>
<td>sp_drop_sql_remote procedure</td>
<td>394</td>
</tr>
<tr>
<td>sp_grant_consolidate procedure</td>
<td>395</td>
</tr>
<tr>
<td>sp_grant_remote procedure</td>
<td>397</td>
</tr>
<tr>
<td>sp_modify_article procedure</td>
<td>399</td>
</tr>
<tr>
<td>sp_modify_remote_table procedure</td>
<td>401</td>
</tr>
<tr>
<td>sp_passthrough procedure</td>
<td>403</td>
</tr>
<tr>
<td>sp_passthrough_piece procedure</td>
<td>404</td>
</tr>
<tr>
<td>sp_passthrough_stop procedure</td>
<td>406</td>
</tr>
<tr>
<td>sp_passthrough_subscription procedure</td>
<td>407</td>
</tr>
<tr>
<td>sp_passthrough_user procedure</td>
<td>408</td>
</tr>
<tr>
<td>sp_populate_sql_anywhere procedure</td>
<td>409</td>
</tr>
<tr>
<td>sp_publisher procedure</td>
<td>410</td>
</tr>
<tr>
<td>sp_queue_clean procedure</td>
<td>411</td>
</tr>
<tr>
<td>sp_queue_delete_old procedure</td>
<td>412</td>
</tr>
<tr>
<td>sp_queue_drop procedure</td>
<td>413</td>
</tr>
</tbody>
</table>
### sp_add_article procedure

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_queue_dump_database procedure</td>
<td>414</td>
</tr>
<tr>
<td>sp_queue_dump_transaction procedure</td>
<td>415</td>
</tr>
<tr>
<td>sp_queue_get_state procedure</td>
<td>416</td>
</tr>
<tr>
<td>sp_queue_read procedure</td>
<td>417</td>
</tr>
<tr>
<td>sp_queue_reset procedure</td>
<td>418</td>
</tr>
<tr>
<td>sp_queue_set_confirm procedure</td>
<td>419</td>
</tr>
<tr>
<td>sp_queue_set_progress procedure</td>
<td>420</td>
</tr>
<tr>
<td>sp_queue_transaction procedure</td>
<td>421</td>
</tr>
<tr>
<td>sp_remote procedure</td>
<td>422</td>
</tr>
<tr>
<td>sp_remote_option procedure</td>
<td>423</td>
</tr>
<tr>
<td>sp_remote_type procedure</td>
<td>424</td>
</tr>
<tr>
<td>sp_remove_article procedure</td>
<td>425</td>
</tr>
<tr>
<td>sp_remove_article_col procedure</td>
<td>426</td>
</tr>
<tr>
<td>sp_remove_remote_table procedure</td>
<td>427</td>
</tr>
<tr>
<td>sp_revoke_consolidate procedure</td>
<td>428</td>
</tr>
<tr>
<td>sp_revoke_remote procedure</td>
<td>429</td>
</tr>
<tr>
<td>sp_subscription procedure</td>
<td>430</td>
</tr>
</tbody>
</table>
sp_add_article procedure

Purpose
To add an article to a publication.

Syntax

```
sp_add_article publication_name, table_name, where_expr,
    subscribe_by_expr
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication to which the article is to be added.</td>
</tr>
<tr>
<td>table_name</td>
<td>The table containing the article.</td>
</tr>
<tr>
<td>where_expr</td>
<td>This optional argument must be a column name or NULL. The publication includes only rows for which the supplied column value is not NULL. The default value is NULL, in which case no rows are excluded from the publication.</td>
</tr>
<tr>
<td>subscribe_by_expr</td>
<td>The new subscription expression defining which rows are to be included in the publication for each subscription. The expression must be the name of a column in table_name. The default value is NULL.</td>
</tr>
</tbody>
</table>

See also
"sp_add_remote_table procedure" on page 389
"sp_create_publication procedure" on page 391
"sp_remove_article procedure" on page 425

Description
Running `sp_add_article` adds an article to a publication. The table must be marked for replication using the "sp_add_remote_table procedure" on page 389 before it can be added to a publication; failure to do so leads to an error.

Calling `sp_add_article` adds all the columns of the table to a publication. If you wish to include only some of the columns of the table in a publication you must first run `sp_add_article` and then call the "sp_add_article_col procedure" on page 388.

As with other data definition changes, in a production environment this procedure should only be run on a quiet SQL Remote installation.

∞ For more information on the requirements for a quiet system, see "Making schema changes" on page 289.

Example

- The following statement adds the SalesRep table to a publication named SalesRepData:

```
sp_add_article 'SalesRepData', 'SalesRep'
```
**sp_add_article_col procedure**

**Purpose**
To add a column to an article in a publication.

**Syntax**
```sql
sp_add_article_col publication_name, table_name, column_name
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>publication_name</code></td>
<td>The name of the publication to which the article is to be added.</td>
</tr>
<tr>
<td><code>table_name</code></td>
<td>The table containing the article.</td>
</tr>
<tr>
<td><code>column_name</code></td>
<td>The column to be added to the article in a publication.</td>
</tr>
</tbody>
</table>

**See also**
"sp_add_article procedure" on page 387
"sp_remove_article procedure" on page 425

**Description**
Running **sp_add_article_col** adds a column to an article in a publication. The table must first be added to the publication using the "sp_add_article procedure" on page 387.

To add all the columns of a table to a publication you do not need to use `sp_add_article_col`; just call `sp_add_article`.

To add only some of the columns of a table to a publication you first call `sp_add_article`, and then call `sp_add_article_col` for each of the columns you wish to include in the publication.

As with other data definition changes, in a production environment this procedure should only be run on a quiet SQL Remote installation.

지도 For more information on the requirements for a quiet system, see "Making schema changes" on page 289.

**Example**

The following statements add the `emp_id` and `emp_name` columns of the `employee` table to a publication named `Personnel`:

```sql
sp_add_article 'Personnel', employee'
sp_add_article_col 'Personnel', 'employee', 'emp_id'
sp_add_article_col 'Personnel', 'employee', 'emp_name'
go
```
sp_add_remote_table procedure

Purpose
To mark a table for SQL Remote replication.

Syntax
```
sp_add_remote_table table_name,
    [ resolve_procedure, ]
    [ old_row_name, ]
    [ remote_row_name ]
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The table to be marked for SQL Remote replication.</td>
</tr>
<tr>
<td>resolve_procedure</td>
<td>The name of a stored procedure that carries out actions when a conflict occurs.</td>
</tr>
<tr>
<td>old_row_name</td>
<td>The name of a table holding the values in the table when a conflict occurs.</td>
</tr>
<tr>
<td>remote_row_name</td>
<td>The name of a table holding the values at the remote database when a conflict-causing UPDATE statement was applied.</td>
</tr>
</tbody>
</table>

Authorization
You must be a system administrator to execute this procedure.

See also
"sp_modify_remote_table procedure" on page 401
"sp_remove_remote_table procedure" on page 427
"Managing conflicts" on page 182.

Description
Each table in a database must be marked for replication by using `sp_add_remote_table` before it can be included in any SQL Remote publications. After executing `sp_add_remote_table`, you can add the table to a publication using the "sp_add_article procedure" on page 387 and the "sp_add_article_col procedure" on page 388.

The `sp_add_remote_table` procedure calls `sp_setreplicate`, which flags the table for replication. This tells Adaptive Server Enterprise to put extended information into the transaction log. This information includes the entire before and after images of the row.

The first argument is the name of the table to be marked for replication.

The remaining three arguments are optional. They are object names required only for custom conflict resolution. If you are implementing custom conflict resolution, you must supply the names of two tables, and a stored procedure. The `sp_add_remote_table` procedure does not check for the existence of the conflict resolution arguments; you can create them either before or after marking the table for replication.

The two tables must have the same columns and data types as table `table_name`. 

389
sp_add_remote_table procedure

Examples

♦️ The following statement marks the Customer table for replication, using default conflict resolution:

```sql
exec sp_add_remote_table Customer
```

♦️ The following statement marks the Customer table for replication, using a stored procedure named Customer_Conflict to resolve conflicts. The old and remote rows are stored in tables named old_Customer and remote_Customer, respectively:

```sql
exec sp_add_remote_table Customer,  
Customer_Conflict, old_Customer, remote_Customer
```
sp_create_publication procedure

Purpose
To create a publication.

Syntax
```
sp_create_publication publication_name
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication</td>
</tr>
</tbody>
</table>

See also
"sp_drop_publication procedure" on page 392

Description
Running sp_create_publication creates a publication, but one with no content. Once the publication is created, you must add articles to it using the "sp_add_remote_table procedure" on page 389 and the "sp_add_article procedure" on page 387.

Example

♦ The following statement creates a publication named SalesRepData:

```
sp_create_publication 'SalesRepData'
go
```
**sp_drop_publication procedure**

**Purpose**
To drop a publication from the database.

**Syntax**
```
sp_drop_publication publication_name
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication to be dropped</td>
</tr>
</tbody>
</table>

**See also**
"sp_create_publication procedure" on page 391

**Description**
Running `sp_drop_publication` drops a publication from the database. All articles that make up the publication, and subscriptions to the publication, are also dropped.

**Example**
♠ The following statement drops the publication named SalesRep:

```
sp_drop_publication 'SalesRep'
go
```
**sp_drop_remote_type procedure**

**Purpose**
To drop a message type from the database.

**Syntax**
```sql
sp_drop_remote_type type_name
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>type_name</code></td>
<td>The message type to drop. This must be a string containing one of the following:</td>
</tr>
<tr>
<td></td>
<td>♦ file</td>
</tr>
<tr>
<td></td>
<td>♦ ftp</td>
</tr>
<tr>
<td></td>
<td>♦ smtp</td>
</tr>
<tr>
<td></td>
<td>♦ mapi</td>
</tr>
<tr>
<td></td>
<td>♦ vim</td>
</tr>
</tbody>
</table>

**See also**
"sp_remote_type procedure" on page 424

**Description**
Drops the named message type from the database.

**Example**
♦ The following statement drops the MAPI message type from the database:

```sql
sp_drop_remote_type mapi
go
```
sp_drop_sql_remote procedure

Purpose
To drop the SQL Remote system objects from a database.

Syntax
sp_drop_sql_remote

See also
"sp_queue_drop procedure" on page 413

Description
Drops the SQL Remote system objects from the database, so that it can no
longer function in a SQL Remote installation.

The sole SQL Remote object not removed is the sp_drop_sql_remote
procedure itself (a procedure cannot drop itself from a database). To
complete removal of SQL Remote requires that sp_drop_sql_remote be
dropped explicitly after it is called.

The sp_drop_sql_remote procedure does not remove stable queue objects
from the database. To remove the stable queue, use the "sp_queue_drop
procedure" on page 413.

Example
◆ The following statements remove SQL Remote system objects from a
database:

    sp_drop_sql_remote_type
    go
    drop procedure sp_drop_sql_remote
    go
sp_grant_consolidate procedure

Purpose
To identify a database immediately above the current database in a SQL Remote hierarchy, who will receive messages from the current database. This procedure applies only to Adaptive Server Enterprise databases acting as remote databases.

Syntax
sp_grant_consolidate user_name, type_name, address

[. frequency ] [. send_time ]

Argument | Description
---|---
user_name | The user ID who will be able to receive SQL Remote messages.
type_name | The message type to drop. This must be one of the following:
- file
- ftp
- smtp
- mapi
- vim
address | A string holding the address, according to the specified message type, to which the replication messages should be sent for this user.
frequency | A string containing one of the following:
- SEND EVERY Indicates that messages are sent at a frequency specified by send_time.
- SEND AT Indicates that messages are sent at a time of day specified by send_time.
send_time | A string containing a time specification with the following meaning:
- If frequency is SEND EVERY, specifies a length of time between messages.
- If frequency is SEND AT, specifies a time of day at which messages will be sent.

If no frequency is specified, the Message Agent sends messages, and then stops.

See also
"sp_grant_remote procedure" on page 397
"sp_revoke_consolidate procedure" on page 428
**sp_grant_consolidate procedure**

**Description**
If the Adaptive Server Enterprise server is acting as a remote database in a SQL Remote installation, the single database above the current database must be granted consolidated permissions using the `sp_grant_consolidate` procedure.

The consolidated user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The address-name must be a valid address for the message-system, enclosed in single quotes.

The `sp_grant_consolidate` procedure is required for the remote database to receive messages, but does not by itself subscribe the remote user to any data. To subscribe to data, a subscription must be created for the user ID to one of the publications in the current database.

The optional `frequency` argument specifies a frequency at which messages are sent. The `send_time` argument contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If no `frequency` argument is supplied, the Message Agent processes messages, and then stops. In order to run the Message Agent continuously, you must ensure that every user with remote or consolidated permission has a frequency specified.

**Example**

- The following statement grants consolidated permissions to user `hq_user`, using a file sharing system, sending messages to the address `hq_dir`: No frequency arguments are specified, and the Message Agent will run in batch mode.

  ```sql
  sp_grant_consolidate
  @user_name=hq_user,
  @address=hq_dir,
  @type_name=file
  go
  ```
sp_grant_remote procedure

Purpose
To identify a database immediately below the current database in a SQL Remote hierarchy, who will receive messages from the current database. These are called remote users.

Syntax
\[ \text{sp_grant_remote user\_name, type\_name, address} \]
\[ [, \text{frequency}] [, \text{send\_time}] \]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>The user ID who will be able to receive SQL Remote messages.</td>
</tr>
<tr>
<td>type_name</td>
<td>The message type to drop. This must be one of the following:</td>
</tr>
<tr>
<td></td>
<td>♦ file</td>
</tr>
<tr>
<td></td>
<td>♦ ftp</td>
</tr>
<tr>
<td></td>
<td>♦ smtp</td>
</tr>
<tr>
<td></td>
<td>♦ mapi</td>
</tr>
<tr>
<td></td>
<td>♦ vim</td>
</tr>
<tr>
<td>address</td>
<td>A string holding the address, according to the specified message type, to which the replication messages should be sent for this user.</td>
</tr>
<tr>
<td>frequency</td>
<td>A string containing one of the following:</td>
</tr>
<tr>
<td></td>
<td>♦ SEND EVERY Indicates that messages are sent at a frequency specified by send_time.</td>
</tr>
<tr>
<td></td>
<td>♦ SEND AT Indicates that messages are sent at a time of day specified by send_time.</td>
</tr>
<tr>
<td>send_time</td>
<td>An optional string containing a time specification with the following meaning:</td>
</tr>
<tr>
<td></td>
<td>♦ If frequency is SEND EVERY, specifies a length of time between messages.</td>
</tr>
<tr>
<td></td>
<td>♦ If frequency is SEND AT, specifies a time of day at which messages will be sent.</td>
</tr>
</tbody>
</table>

If no frequency is specified, the Message Agent sends messages, and then stops.

See also
"sp_revoke_remote procedure" on page 429
**sp_grant_remote procedure**

**Description**

In a SQL Remote installation, each database receiving messages from the current database must be granted REMOTE permissions using the `sp_grant_remote` procedure.

The remote user is identified by a message system, identifying the method by which messages are sent to and received from the consolidated user. The address-name must be a valid address for the message-system, enclosed in single quotes.

The `sp_grant_remote` procedure is required for the remote database to receive messages, but does not by itself subscribe the remote user to any data. To subscribe to data, a subscription must be created for the user ID to one of the publications in the current database.

The optional *frequency* argument specifies a frequency at which messages are sent. The *send time* argument contains a time that is a length of time between messages (for SEND EVERY) or a time of day at which messages are sent (for SEND AT). With SEND AT, messages are sent once per day.

If no *frequency* argument is supplied, the Message Agent processes messages, and then stops. In order to run the Message Agent continuously, you must ensure that every user with REMOTE permission has a frequency specified.

It is anticipated that at many consolidated databases, the Message Agent will be run continuously, so that all remote databases would have a *frequency* argument specified. A typical setup may involve sending messages to laptop users daily (SEND AT) and to remote servers every hour or two (SEND EVERY). You should use as few different times as possible, for efficiency.

**Example**

- The following statement grants remote permissions to user **SamS**, using a MAPI e-mail system, sending messages to the address **Singer, Samuel** once every two hours:

  ```sql
  exec sp_grant_remote 'SamS',
                        'mapi',
                        'Singer, Samuel',
                        'SEND EVERY',
                        '02:00'
  go
  ```
sp_modify_article procedure

Purpose
To change the description of an article in a procedure.

Syntax
sp_modify_article
   publication_name,
   table_name,
   [ where_expr, ]
   [ subscribe_by_expr ]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication for which the article is to be modified.</td>
</tr>
<tr>
<td>table_name</td>
<td>The table containing the article.</td>
</tr>
<tr>
<td>where_expr</td>
<td>This optional argument must be a column name or NULL. The publication includes only rows for which the supplied column name is not NULL. The default value is NULL, in which case no rows are excluded from the publication.</td>
</tr>
<tr>
<td>subscribe_by_expr</td>
<td>The new subscription expression defining which rows are to be included in the publication for each subscription. The default value is NULL.</td>
</tr>
</tbody>
</table>

See also
"sp_add_article procedure" on page 387
"sp_remove_article procedure" on page 425

Description
To change the description of an article in a publication. The WHERE expression and the subscription expression can both be changed.

As with other data definition changes, in a production environment this procedure should only be run on a quiet SQL Remote installation.

For more information on the requirements for a quiet system, see "Making schema changes" on page 289.

Examples

♦ The following statement changes an article in the SalesRepData publication that takes information from the Customer table, so that it has no subscription expression:

   sp_modify_article SalesRepData, Customer
go

♦ The following statement changes an article in the SalesRepData publication that takes information from the Customer table, so that it has a subscription expression that is the rep_key column:

   sp_modify_article SalesRepData,
sp_modify_article procedure

    Customer, NULL, rep_key
    go
sp_modify_remote_table procedure

Purpose
To change the resolution objects for a table marked for SQL Remote replication.

Syntax
```
sp_modify_remote_table table_name,
  [ resolve_name, ]
  [ old_row_name, ]
  [ remote_row_name ]
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>A table marked for SQL Remote replication.</td>
</tr>
<tr>
<td>resolve_procedure</td>
<td>The name of the new stored procedure for carrying out actions when a conflict occurs.</td>
</tr>
<tr>
<td>old_row_name</td>
<td>The name of the new table for holding the values in the table when a conflict occurs.</td>
</tr>
<tr>
<td>remote_row_name</td>
<td>The name of the new table for holding the values at the remote database when a conflict-causing UPDATE statement was applied.</td>
</tr>
</tbody>
</table>

See also
- "sp_add_remote_table procedure" on page 389
- "sp_remove_remote_table procedure" on page 427
- "Managing conflicts" on page 182.

Description
Each table in a database must be marked for replication by using `sp_add_remote_table` before it can be included in any SQL Remote publications.

The `sp_modify_remote_table` allows you to change the way in which conflict resolution is carried out for update conflicts occurring on this table.

The arguments are, in addition to the table name, the object names required for custom conflict resolution. If you are implementing custom conflict resolution, you must supply the names of two tables, and a stored procedure. The `sp_modify_remote_table` procedure does not check for the existence of the conflict resolution arguments: you can create them either before or after marking the table for replication.

The two tables must have the same columns and data types as table `table_name`.

Example

- The following statement instructs SQL Remote to use the `resolve_Cust` procedure, the `old_Cust` table, and the `remote_Cust` table to resolve update conflicts on the `Customer` table:

  ```
  sp_add_remote_table Customer, 
  resolve_Cust,
  ```
sp_modify_remote_table procedure

    old_Cust,
    remote_Cust
    go
sp_passthrough procedure

Purpose
To send a SQL statement in passthrough mode.

Syntax
```
sp_passthrough statement
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement</td>
<td>A string containing a statement to be executed in passthrough mode.</td>
</tr>
</tbody>
</table>

See also
"sp_passthrough_piece procedure" on page 404
"sp_passthrough_stop procedure" on page 406
"sp_passthrough_subscription procedure" on page 407
"sp_passthrough_user procedure" on page 408

Description
To send passthrough operations. The recipients of the passthrough statement are determined by previous calls to `sp_passthrough_user` and `sp_passthrough_subscription`.

The string must be less than 255 characters long. For SQL statements longer than 255 characters, you should execute a sequence of calls to the `sp_passthrough_piece` procedures, and execute `sp_passthrough` for the final piece of the statement and to cause the replication to occur.

Caution
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.

Example
- The following statement sends a create table statement to the current recipients of passthrough statements.

```
exec sp_passthrough
  "CREATE TABLE simple (id integer NOT NULL, name char(50))"
go
```
sp_passthrough Piece procedure

Purpose
To build a long SQL statement for passthrough.

Syntax
sp_passthrough_piece string

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>A piece of a statement to be executed in passthrough mode.</td>
</tr>
</tbody>
</table>

See also
"sp_passthrough procedure" on page 403
"sp_passthrough_stop procedure" on page 406
"sp_passthrough_subscription procedure" on page 407
"sp_passthrough_user procedure" on page 408

Description
The "sp_passthrough procedure" on page 403 is used to send statements directly to a set of remote users. Statements that are longer than 255 characters have to be built up piece by piece.

To build and send a long SQL statement, call sp_passthrough_piece for all but the final piece of the statement, and then call sp_passthrough for the final piece. This completes and replicates the statement.

All pieces of a passthrough statement must be built within a single transaction.

Example

♦ The following statements send a long passthrough statement to the current list of passthrough recipients:

```
begin transaction
    go
    exec sp_passthrough_piece 'CREATE TABLE DBA.employee
      (emp_id integer NOT NULL,
       manager_id integer NULL,
       emp_fname char(20) NOT NULL,
       emp_lname char(20) NOT NULL,
     )
    go
    exec sp_passthrough_piece 'CREATE TABLE DBA.employee
      (dept_id integer NOT NULL,
       street char(40) NOT NULL,
       city char(20) NOT NULL,
       state char(4) NOT NULL,
       zip_code char(9) NOT NULL,
       phone char(10) NULL,
     )
    go
    exec sp_passthrough_piece 'CREATE TABLE DBA.employee
      (status char(1) NULL,
       ss_number char(11) NOT NULL,
       salary numeric(20,3) NOT NULL,
     )
```
start_date date NOT NULL,
termination_date date NULL,
birth_date date NULL,

GO
EXEC sp_passthrough '
  bene_health_ins char(1) NULL,
bene_life_ins char(1) NULL,
bene_day_care char(1) NULL,
sex char(1) NULL,
PRIMARY KEY (emp_id),

)

GO
commit
GO
**sp_passthrough_stop procedure**

**Purpose**  
Resets passthrough mode

**Syntax**  
`sp_passthrough_stop`

**See also**  
"sp_passthrough procedure" on page 403  
"sp_passthrough_subscription procedure" on page 407  
"sp_passthrough_user procedure" on page 408

**Description**  
The `sp_passthrough_stop` procedure resets the list of recipients of passthrough statements to be empty, and clears any statements that are currently being built.

**Example**  
♦ The following statement resets the passthrough recipient list to be empty.

```sql
exec sp_passthrough_stop  
go
```
**sp_passthrough_subscription procedure**

**Purpose**
Adds subscribers to a given publication to the recipient list for passthrough statements.

**Syntax**
```
sp_passthrough_subscription publication_name, subscribe_by
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication</td>
</tr>
<tr>
<td>subscribe_by</td>
<td>The subscription value for recipients to receive passthrough statements.</td>
</tr>
</tbody>
</table>

**See also**
"sp_passthrough procedure" on page 403
"sp_passthrough_piece procedure" on page 404
"sp_passthrough_stop procedure" on page 406
"sp_passthrough_user procedure" on page 408

**Description**
This is one of two ways that you can add to the list of recipients for passthrough statements, the other being to use the "sp_passthrough_user procedure" on page 408.

The users that are added to the recipient list by a call to the `sp_passthrough_subscription` procedure are all those users subscribing to the publication `publication_name` with a subscription value of `subscribe_by`.

The default setting for `subscribe_by` is NULL. In this case, all subscribers to the publication receive the passthrough statements.

**Example**
- The following statement adds to the list of passthrough recipients the subscriber or subscribers to the `SalesRepData` publication who use subscription values of 'rep1'.

  ```
  Sp_passthrough_subscription SalesRepData, rep1
  ```
sp_passthrough_user procedure

Purpose
Adds a named user to the list of recipients for passthrough statements.

Syntax
\texttt{sp\_passthrough\_user user\_name}

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>The user to be added to the list of recipients.</td>
</tr>
</tbody>
</table>

See also
"sp\_passthrough\_procedure" on page 403
"sp\_passthrough\_piece procedure" on page 404
"sp\_passthrough\_stop procedure" on page 406
"sp\_passthrough\_subscription procedure" on page 407

Description
This is one of two ways that you can add to the list of recipients for passthrough statements, the other being to use the "sp\_passthrough\_subscription procedure" on page 407.

The sp\_passthrough\_user procedure adds the named user to the list of recipients for passthrough statements. The list remains in force until reset using the "sp\_passthrough\_stop procedure" on page 406.

Example
\begin{itemize}
  \item The following statement adds the user field\_user to the list of recipients for passthrough statements:
  \begin{verbatim}
  sp\_passthrough\_user 'field\_user'
go
  \end{verbatim}
\end{itemize}
sp_populate_sql_anywhere procedure

Purpose
To create a copy of the Adaptive Server Anywhere system tables in the TEMPDB. This procedure is used by the extraction utility ssxtract.

Syntax
sp_populate_sql_anywhere

Description
To create a set of Adaptive Server Anywhere system tables for a remote Adaptive Server Anywhere database, in TEMPDB. The information is used by the extraction utility to construct an Adaptive Server Anywhere database schema from the set of publications in the Adaptive Server Enterprise consolidated database.

This procedure is used by the ssxtract extraction utility. It should not be called directly.
sp_publisher procedure

Purpose
To set the publisher of the current database, or to remove the publisher.

Syntax
sp_publisher [ user_name ]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>The user ID to be identifies as the publisher for the database.</td>
</tr>
</tbody>
</table>

See also
"Managing SQL Remote permissions" on page 219.

Description
Each database in a SQL Remote installation is identified in outgoing messages by a user ID, called the publisher. The sp_publisher procedure sets the publisher user ID associated with these outgoing messages.

Each database can have at most one publisher; if a publisher already exists, sp_publisher changes the name of the publisher.

If no user_name argument is provided, the current publisher is removed, so that the database has no publisher. Only the permission to be the publisher is removed; the user ID is not removed from the database.

Examples
♦ The following statement identifies the user ID joe as the publisher of the current database:

```
sp_publisher joe

```

♦ The following statement sets the current database to have no publisher:

```
sp_publisher

```
sp_queue_clean procedure

<table>
<thead>
<tr>
<th>Purpose</th>
<th>This procedure is used by the SQL Remote Message Agent, and should not be called directly.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>sp_queue_clean</td>
</tr>
<tr>
<td>Description</td>
<td>This procedure is used by the SQL Remote Message Agent, and should not be called directly. It removes from the stable queue any transactions that completed after the start of the oldest incomplete transaction the last time the log was scanned.</td>
</tr>
</tbody>
</table>
sp_queue_delete_old procedure

Purpose
This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax
sp_queue_delete_old

Description
This procedure is used by the SQL Remote Message Agent, and should not be called directly. It deletes from the stable queue any transactions that have been confirmed by all remote databases.
sp_queue_drop procedure

Purpose
To drop the stable queue objects from a database.

Syntax
sp_queue_drop

See also
"sp_drop_sql_remote procedure" on page 394

Description
Drops the stable queue system objects from the database, so that the database no longer supports a SQL Remote stable queue.

The sole stable queue object not removed is the sp_queue_drop procedure itself (a procedure cannot drop itself from a database). To complete removal of the stable queue requires that sp_queue_drop be dropped explicitly after it is called.

The sp_queue_drop procedure does not remove SQL Remote system objects from the database. To remove the SQL Remote system objects, use the "sp_drop_sql_remote procedure" on page 394.

Examples
♦ The following statements remove the stable queue objects from the database:

```
sp_queue_drop
go
drop procedure sp_queue_drop
go
```
sp_queue_dump_database procedure

Purpose
To facilitate recovery from media failure when the stable queue is in a separate database from the SQL Remote objects.

Syntax
sp_queue_dump_database

See also
"sp_queue_dump_transaction procedure" on page 415
"Stable queue recovery issues" on page 287

Description
Keeping the stable queue in a separate database complicates backup and recovery, as consistent versions of the two databases have to be recovered.

Normal recovery automatically restores the two databases to a consistent state, but recovery from media failure takes some care. When restoring database dumps, it is important to recover the stable queue to a consistent point. The sp_queue_dump_database procedure is provided to help with recovery from media failure. It is called whenever a dump database is scanned.

As provided, the procedure does not carry out any operations. You can modify this stored procedure to issue a dump database command in the stable store database.
sp_queue_dump_transaction procedure

Purpose
To facilitate recovery from media failure, when the stable queue is in a separate database from the SQL Remote objects.

Syntax
sp_queue_dump_transaction

See also
"sp_queue_dump_database procedure" on page 414
"Stable queue recovery issues" on page 287

Description
Keeping the stable queue in a separate database complicates backup and recovery, as consistent versions of the two databases have to be recovered.

Normal recovery automatically restores the two databases to a consistent state, but recovery from media failure takes some care. When restoring database dumps, it is important to recover the stable queue to a consistent point. The sp_queue_dump_transaction procedure is provided to help with recovery from media failure. It is called whenever a dump transaction is scanned.

As provided, the procedure does not carry out any operations. You can modify this stored procedure to issue a dump transaction command in the stable store database.
sp_queue_get_state procedure

Purpose
This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax
sp_queue_get_state

Description
This procedure is used by the SQL Remote Message Agent, and should not be called directly. It returns a description of the current state of the stable queue.
sp_queue_read procedure

**Purpose**
This procedure is used by the SQL Remote Message Agent, and should not be called directly.

**Syntax**
*sp_queue_read* start_offset, stop_offset

**Description**
This procedure reads transactions from the stable queue. It is exclusively for use by the Message Agent.
sp_queue_reset procedure

Purpose
To reset the server to a point where the stable queue is empty.

Syntax
sp_queue_reset

Description
This procedure is used by the SQL Remote Message Agent, and should not be called directly in a production environment. It deletes all rows from the stable queue sr_transaction table, and resets the sr_queue_state table, ready for a new SQL Remote setup.

In a development phase, this procedure can be useful to reset the server.
sp_queue_set_confirm procedure

Purpose
This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax
sp_queue_set_confirm confirm_offset

Description
This procedure is used by the SQL Remote Message Agent, and should not be called directly. It sets the minimum confirmation offset from all remote users in the sr_queue_state table.
sp_queue_set_progress procedure

Purpose
This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax
sp_queue_set_progress page_id, row_id, commit_offset, backup_offset, marker

Description
This procedure is used by the SQL Remote Message Agent, and should not be called directly. It sets the transaction log scanning progress value in the sr_queue_state table.
sp_queue_transaction procedure

Purpose
This procedure is used by the SQL Remote Message Agent, and should not be called directly.

Syntax
sp_queue_transaction offset, user_id

Description
This procedure is used by the SQL Remote Message Agent, and should not be called directly. It adds a new transaction to the stable queue.
**sp_remote procedure**

**Purpose**
This procedure is used by the SQL Remote Message Agent, and should not be called directly, with a single exception described below. It manages rows in the `sr_remoteuser` table.

**Syntax**

```
sp_remote operation, user_name [, offset ] [, confirm ]
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation</td>
<td>The name of an action. The only value that should be used by a user is reset; all others are for use by the Message Agent.</td>
</tr>
<tr>
<td>user_name</td>
<td>The name of the remote user being reset</td>
</tr>
<tr>
<td>offset</td>
<td>Not used</td>
</tr>
<tr>
<td>confirm</td>
<td>Not used</td>
</tr>
</tbody>
</table>

**Description**
This procedure is used by the SQL Remote Message Agent, and should not be called directly with the single exception of the reset call. It maintains the message tracking information in the `sr_remoteuser` table.

The following special case can be used directly, when creating a custom database extraction process:

```
sp_remote reset, remote_user
```

where `remote_user` is the remote user name.

This command starts all subscriptions for a remote user in a single transaction. It sets the `log_sent` and `confirm_sent` values in `sr_remoteuser` table to the current position in the transaction log. It also sets the created and started values in `sr_subscription` to the current position in the transaction log for all subscriptions for this remote user. The procedure does not do a commit. You must do an explicit commit after this call.

In order to write an extraction process that is safe on a live database, the data must be extracted at isolation level 3 in the same transaction as the subscriptions are started.
sp_remote_option procedure

**Purpose**
To set a SQL Remote option.

**Syntax**
```plaintext
sp_remote_option option_name, option_value
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option_name</td>
<td>The name of one of the SQL Remote options</td>
</tr>
<tr>
<td>option_value</td>
<td>The value to which the option is set.</td>
</tr>
</tbody>
</table>

**See also**
"SQL Remote options" on page 323.

**Description**
The SQL Remote options provide control over replication behavior. The following options are available in Adaptive Server Enterprise:

<table>
<thead>
<tr>
<th>OPTION</th>
<th>VALUES</th>
<th>DEFAULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob_threshold</td>
<td>Integer, in K</td>
<td>256</td>
</tr>
<tr>
<td>Compression</td>
<td>-1 to 9</td>
<td>6</td>
</tr>
<tr>
<td>Delete_old_logs</td>
<td>ON, OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>Qualify_owners</td>
<td>ON, OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>Quote_all_identifiers</td>
<td>ON, OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>Replication_error</td>
<td>procedure-name</td>
<td>NULL</td>
</tr>
<tr>
<td>Subscribe_by_remote</td>
<td>ON, OFF</td>
<td>ON</td>
</tr>
<tr>
<td>Verify_threshold</td>
<td>integer</td>
<td>256</td>
</tr>
<tr>
<td>Verify_all_columns</td>
<td>ON, OFF</td>
<td>OFF</td>
</tr>
</tbody>
</table>

A complete description of these options is provided in "SQL Remote options" on page 323.

**Example**
- The following statement sets the Verify_all_columns option to OFF, so that old values of update statements applied by the Message Agent are not checked automatically for all columns.

  ```plaintext
  sp_remote_option Verify_all_columns, OFF
go
  ```
sp_remote_type procedure

Purpose
To create or modify a SQL Remote message type.

Syntax
sp_remote_type type_name publisher_address

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type_name</td>
<td>The message type to create or alter. This must be one of the following:</td>
</tr>
<tr>
<td></td>
<td>♦ file</td>
</tr>
<tr>
<td></td>
<td>♦ ftp</td>
</tr>
<tr>
<td></td>
<td>♦ smtp</td>
</tr>
<tr>
<td></td>
<td>♦ mapi</td>
</tr>
<tr>
<td></td>
<td>♦ vim</td>
</tr>
<tr>
<td>publisher_address</td>
<td>The address of the publisher under the specified message type.</td>
</tr>
</tbody>
</table>

See also
"sp_drop_remote_type procedure" on page 393

Description
The Message Agent sends outgoing messages from a database using one of the supported message links. Return messages for users employing the specified link are sent to the specified address as long as the remote database is created by the Extraction Utility. The Message Agent starts links only if it has remote users for those links.

The address is the publisher's address under the specified message system. If it is an e-mail system, the address string must be a valid e-mail address. If it is a file-sharing system, the address string is a subdirectory of the directory set in the SQLREMOTE environment variable or registry entry, or of the current directory if that is not set.

For the FILE link, the procedure also causes the Message Agent to look for incoming messages in the address given for each message type.

Example
♦ The following example creates a FILE message type for a database, and gives the publisher's address as a subdirectory of the SQLREMOTE location named publisher:

    sp_remote_type file, publisher

    go
sp_remove_article procedure

Purpose
To remove an article from a publication

Syntax
sp_remove_article publication_name, table_name

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication from which the article is to be deleted.</td>
</tr>
<tr>
<td>table_name</td>
<td>The table containing the article.</td>
</tr>
</tbody>
</table>

See also
"sp_add_article procedure" on page 387

Description
Running sp_add_article removes an article from a publication. Any article including parts of the named table is removed from the publication.

Example
♦ The following statement removes any articles that use part of the SalesRep table from a publication named SalesRepData:

```
sp_remove_article SalesRepData, SalesRep
go
```
sp_remove_article_col procedure

**Purpose**
To remove a column from an article in a publication.

**Syntax**
```
sp_remove_article_col publication_name, article_name, column_name
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>publication_name</td>
<td>The name of the publication to which the article belongs.</td>
</tr>
<tr>
<td>article_name</td>
<td>The article from which the column is to be removed.</td>
</tr>
<tr>
<td>column_name</td>
<td>The column to be removed from the article.</td>
</tr>
</tbody>
</table>

**See also**
"sp_add_article_col procedure" on page 388
"sp_remove_article procedure" on page 425

**Description**
You can remove a column from a publication using `sp_remove_article_col`.

To remove a column using `sp_remove_article_col`, the column must have been explicitly added to a publication using the "sp_add_article_col procedure" on page 388. Although the "sp_add_article procedure" on page 387, without use of `sp_add_article_col`, adds all the columns of a table to a publication, you cannot remove a single column from such a publication using `sp_remove_article_col`.

**Example**
♦ The following statement removes the column `emp_lname` of the `employee` table from a publication named `Personnel`:
```
sp_remove_article_col 'Personnel', 'employee', 'emp_lname'
go
```
sp_remove_remote_table procedure

Purpose
To mark a table as unavailable for SQL Remote replication.

Syntax
sp_remove_remote_table table_name

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_name</td>
<td>The table to be marked as not available for SQL Remote replication.</td>
</tr>
</tbody>
</table>

See also
"sp_add_remote_table procedure" on page 389
"sp_modify_remote_table procedure" on page 401

Description
Marks a table as unavailable for replication. Once this procedure has been called, the data in the table cannot be shared with other databases using SQL Remote.

Example
♦ The following statement marks the employee table as unavailable for replication:

    sp_remove_remote_table employee
go
sp_revoke Consolidate Procedure

Purpose
To stop a user from being able to receive SQL Remote messages from this database.

Syntax
sp_revoke Consolidate user_name

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_name</td>
<td>The user ID who will no longer be able to act as a consolidated database.</td>
</tr>
</tbody>
</table>

See also
"sp_grant Consolidate Procedure" on page 395

Description
REMOTE permissions are required for a user ID to receive messages in a SQL Remote replication installation. The sp_revoke Consolidate procedure removes the consolidated database user ID from the list of users receiving messages from the current database.

Example
♦ The following statement revokes consolidated permissions from user hq_user:

```
sp_revoke Consolidate hq_user
```

go
sp_revoke_remote procedure

Purpose
To stop a user from being able to receive SQL Remote messages from this database.

Syntax
```
sp_revoke_remote user_name
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user_name</code></td>
<td>The user ID who will no longer be able to receive SQL Remote messages.</td>
</tr>
</tbody>
</table>

See also
"sp_grant_remote procedure" on page 397

Description
REMOTE permissions are required for a user ID to receive messages in a SQL Remote replication installation. The `sp_revoke_remote` procedure removes a user ID from the list of users receiving messages from the current database.

Example
♦ The following statement revokes remote permissions from user Field User:

```
sp_revoke_remote 'Field user'
go
```
**sp_subscription procedure**

**Purpose**
To manage subscriptions.

**Syntax**
```
sp_subscription operation,
    publication_name,
    user_name,
    [ subscribe_by ]
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operation</code></td>
<td>The operation to be performed. This must be one of the following:</td>
</tr>
<tr>
<td></td>
<td>✷ <code>create</code> To create a subscription to a given publication for a user.</td>
</tr>
<tr>
<td></td>
<td>✷ <code>drop</code> To drop a subscription to a given publication for a user.</td>
</tr>
<tr>
<td></td>
<td>✷ <code>start</code> To start a subscription to the named publication.</td>
</tr>
<tr>
<td></td>
<td>✷ <code>stop</code> To stop a subscription to the named publication.</td>
</tr>
<tr>
<td></td>
<td>✷ <code>synchronize</code> To synchronize a subscription to the named publication.</td>
</tr>
<tr>
<td><code>publication_name</code></td>
<td>The name of the publication to which the subscription refers.</td>
</tr>
<tr>
<td><code>user_name</code></td>
<td>The user ID who is being subscribed to the publication.</td>
</tr>
<tr>
<td><code>subscribe_by</code></td>
<td>The subscription value.</td>
</tr>
</tbody>
</table>

**See also**
"Creating subscriptions" on page 198.

**Description**
The `sp_subscription` procedure is used to manage subscriptions. The first argument to the procedure (`operation`) specified whether the procedure is being created, dropped, started, stopped, or synchronized.

In general, starting and synchronizing subscriptions is done using the extraction utility. Sybase Central does not display subscriptions as started until the Message Agent first runs against the database.

**Example**

- The following statement creates a subscription for user `SalesRep1` to the `SalesRepData` publication, which has no subscription expression.

```
sp_subscription create,
    SalesRepData,
    SalesRep1
go
```
The appendix provides additional information that is not necessarily required for everyday use of the application.
Appendix A

Enterprise and Anywhere: Differences

About this Appendix

This appendix summarizes the differences between SQL Remote for Adaptive Server Enterprise and for Adaptive Server Anywhere.

This appendix describes the main differences between these versions of the technology.

Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types of difference</td>
<td>434</td>
</tr>
<tr>
<td>Differences in functionality</td>
<td>435</td>
</tr>
<tr>
<td>Differences in approach</td>
<td>436</td>
</tr>
<tr>
<td>Limitations for Enterprise to Enterprise replication</td>
<td>438</td>
</tr>
</tbody>
</table>
Types of difference

The differences between the versions of the software are of the following kinds:

♦ **Functionality**  Tasks that can be carried out by one of the two versions, but not by the other.

♦ **Approach**  Although a similar result can be obtained, a different approach is required in each version. This includes tasks that are carried out in ways that are superficially different, but which have the same result.

♦ **Server differences**  Tasks associated with SQL Remote, such as backup management, are different for the two servers. These differences are not described here.

This appendix addresses only replication using Adaptive Server Anywhere as remote databases. There are additional limitations if using Adaptive Server Enterprise as remote servers.
Differences in functionality

The major differences in functionality between SQL Remote for Adaptive Server Enterprise (SRE) and SQL Remote for Adaptive Server Anywhere (SRA) are as follows:

- **Long data types** SRE does not support replication of long data types such as `text` or `image` (also called BLOBS).
- **Schema changes** For SRE, schema changes must be made on a quiet system. A quiet system means the following:
  - **No transactions being replicated** There can be no transactions being replicated that modify the tables that are to be altered. All transactions that modified tables being altered must be scanned from the transaction log into the stable queue before the schema is altered. This is performed by running the Message Agent normally, or using the `-i` `-b` switches. After the Message Agent completes, you can make the schema change.
  - **Message Agent shut down** The Message Agent must be shut down when the schema change is being made.
  - **SQL Remote Open Server** If you are using the SQL Remote Open Server, it must be shut down when the schema change is being made.
- **Trigger action replication** In SRE, trigger actions are replicated. In SRA you have the choice of replicating trigger actions, but by default they are not replicated. The replication of trigger actions requires SRE users to ensure that triggers are not fired at remote databases.
- **Platform availability** SRA is available on a wider variety of platforms that SRE, reflecting the platform availability of the two servers.
- **Publication definitions** Publications in SRA can be more selective than those in SRE. For example, in SRA you can use a `WHERE` clause with any value. In SRE, you can only use `IS NULL` and `IS NOT NULL` conditions in the `WHERE` clause.
Differences in approach

There are some features of SQL Remote that must be approached in a different manner in SRE and SRA.

- **Partitioning tables that do not contain the subscription expression**
  In SRA, publications can contain subqueries, and these allow tables that do not contain a partition expression to nevertheless be distributed properly among subscribers. In SRE, an additional column must be added to such tables, containing a list of subscribers, and triggers must be written to maintain the column. This column can have a maximum size of 255.

  For descriptions, see the section entitled "Partitioning tables that do not contain the subscription expression", in the chapters "SQL Remote Design for Adaptive Server Anywhere", and "SQL Remote Design for Adaptive Server Enterprise", of the book *Data Replication with SQL Remote*.

- **Conflict resolution**
  In SRA, conflict resolution is carried out using a special trigger syntax. In SRE, stored procedures must be written to carry out this task.

  For descriptions, see the section entitled "Managing conflicts", in the chapters "SQL Remote Design for Adaptive Server Anywhere", and "SQL Remote Design for Adaptive Server Enterprise", of the book *Data Replication with SQL Remote*.

- **Storing messages before sending**
  In SRE, a separate table named the **stable queue** is used to hold changes before replication. In SRA, there is no stable queue; instead, the messages are retrieved from current and old transaction log files.

- **Commands**
  Whereas SQL Remote tasks such as creating publications are carried out using SQL statements in SRA, they are carried out using system stored procedures in SRE.

The Sybase Central administration tool hides many of these stylistic differences by providing a common look and feel to the administration of each version of SQL Remote.
Adaptive Server Enterprise procedures and Adaptive Server Anywhere statements

In SQL Remote for Adaptive Server Anywhere, SQL statements are used to carry out the tasks that these stored procedures carry out in Adaptive Server Enterprise. The following table lists the SQL Remote procedures, and how they correspond to SQL statements in Adaptive Server Anywhere:

<table>
<thead>
<tr>
<th>Adaptive Server Enterprise procedure</th>
<th>Corresponding Adaptive Server Anywhere statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_remote_type</td>
<td>CREATE REMOTE MESSAGE TYPE</td>
</tr>
<tr>
<td>sp_remote_type</td>
<td>ALTER REMOTE MESSAGE TYPE</td>
</tr>
<tr>
<td>sp_drop_remote_type</td>
<td>DROP REMOTE MESSAGE TYPE</td>
</tr>
<tr>
<td>sp_grant_remote</td>
<td>GRANT REMOTE</td>
</tr>
<tr>
<td>sp_revoke_remote</td>
<td>REVOKE REMOTE</td>
</tr>
<tr>
<td>sp_publisher</td>
<td>GRANT PUBLISH</td>
</tr>
<tr>
<td>sp_publisher</td>
<td>REVOKE PUBLISH</td>
</tr>
<tr>
<td>sp_create_publication</td>
<td>CREATE PUBLICATION</td>
</tr>
<tr>
<td>sp_add_article</td>
<td>ALTER PUBLICATION</td>
</tr>
<tr>
<td>sp_add_article_col</td>
<td></td>
</tr>
<tr>
<td>sp_add_article</td>
<td>DROP PUBLICATION</td>
</tr>
<tr>
<td>sp_remove_article</td>
<td>CREATE SUBSCRIPTION</td>
</tr>
<tr>
<td>sp_remove_article_col</td>
<td>DROP SUBSCRIPTION</td>
</tr>
<tr>
<td>sp_add_article_col</td>
<td>START SUBSCRIPTION</td>
</tr>
<tr>
<td>sp_drop_publication</td>
<td>STOP SUBSCRIPTION</td>
</tr>
<tr>
<td>sp_subscription 'create'</td>
<td>SYNCHRONIZE SUBSCRIPTION</td>
</tr>
<tr>
<td>sp_subscription 'drop'</td>
<td>PASSTHROUGH FOR USERID</td>
</tr>
<tr>
<td>sp_subscription 'start'</td>
<td>PASSTHROUGH FOR SUBSCRIPTION</td>
</tr>
<tr>
<td>sp_subscription 'stop'</td>
<td>PASSTHROUGH STOP</td>
</tr>
<tr>
<td>sp_subscription 'synchronize'</td>
<td></td>
</tr>
<tr>
<td>sp_passthrough_user</td>
<td></td>
</tr>
<tr>
<td>sp_passthrough_subscription</td>
<td></td>
</tr>
<tr>
<td>sp_passthrough_stop</td>
<td></td>
</tr>
</tbody>
</table>
Limitations for Enterprise to Enterprise replication

If you wish to use SQL Remote for replication between Adaptive Server Enterprise databases, rather than with Adaptive Server Anywhere remote databases, you should be aware of the following limitations:

- **Database extraction** The extraction utility creates RELOAD.SQL scripts and data files for building Adaptive Server Anywhere remote databases. Setting up remote ASE databases requires an extraction process created by the customer.

  ☞ For more information about how to create an extraction process, see "sp_remote procedure" on page 422.

- **Referential integrity errors** Referential integrity is always checked immediately in Adaptive Server Enterprise, while Adaptive Server Anywhere provides the WAIT_FOR_COMMIT option to control when integrity is checked. This presents difficulties when rows move between remote databases, as in territory realignment.

  For example, suppose an Order table has a foreign key to a Customer table which has a foreign key to a SalesRep table. The Customer table is subscribed by sales rep. The Order table is also subscribed by sales rep (it has a redundant column maintained by a trigger).

  When a row in Customer is updated to point to a new sales rep, a trigger fires to update the sales rep column in Order. The update on Customer is replicated as a delete to the old rep and an insert to the new rep. Similarly, the triggered update on Order is replicated as a delete to the old rep and an insert to the new rep.

  The problem occurs because SQL Remote replicates the operations in the order they occur, which means the Customer row is deleted before the Order rows. This causes a referential integrity error.

- **Schema upgrades** Schema upgrades are difficult to manage when both consolidated and remote databases are Adaptive Server Enterprise databases. Passthrough to remote ASE databases is difficult to carry out.

  The problem is due to the need for a quiet system for schema upgrades (see "Differences in functionality" on page 435). Passthrough puts schema upgrade statements into the normal message stream. The operations that precede the schema upgrade (in the same message or a previous message) cannot possibly have been scanned from the transaction log into the stable queue before the schema change takes place.
Appendix A  Enterprise and Anywhere: Differences

- **Synchronize subscription**  This is not implemented for Adaptive Server Enterprise remote databases.
Limitations for Enterprise to Enterprise replication
Index

A
Adaptive Server Anywhere
  creating an Enterprise-compatible database, 96
  address
    setting for publisher, 229
addresses
  file sharing, 232
  ftp, 233
  SMTP, 233
  SMTP/POP, 234
administering
  a SQL Remote installation, 217, 218
administering SQL Remote, 22, 255
administration
  overview, 218
ALTER PUBLICATION statement
  syntax, 353
ALTER REMOTE MESSAGE TYPE statement
  syntax, 354
altering
  message types, 229, 230
  publications, 353
article SQL Remote table
  about, 328, 336
articlecol SQL Remote table
  about, 328, 337
articlecols SQL Remote view
  about, 332, 343
articles
  about, 123, 167
  column-wise partitioning, 162
  notes on, 165
  row-wise partitioning, 163
  system table for, 328, 337
  valid, 123, 167
  whole table, 161
articles SQL Remote view
  about, 332, 343

asademo.db file
  about, xiv

B
b option
  Message Agent, 240
backups
  for remote databases, 271
  for replication, 262, 266, 286
SQL Remote, 241
batch mode
  Message Agent, 239
batches
  passthrough mode, 275
blobs
  replication of, 104

C
cache
  for messages, 244
ccMail
  SQL Remote, 228
collations
  SQL Remote, 97
command-line
  configuration file, 307
  environment variables, 308
Message Agent, 307, 308
commit
  replication, 100
compatibility
Adaptive Server Anywhere Adaptive Server Enterprise, 214, 215
among databases, 96
between Adaptive Server Enterprise Adaptive Server Anywhere, 96
compression
messages, 323
COMPRESSION replication option, 323
configuration file
Message Agent options, 307
conflict detection
about, 101, 143, 183
long data types, 105
conflict resolution
#remote, 187
approaches to, 146, 147, 149
example, 185, 187
implementing, 144, 183
limitations, 185
triggers, 144, 146, 183
conflicts
#remote, 187
about, 110, 142, 143, 182, 183
approaches to resolving, 146, 147, 149
avoiding, 109
detection, 110
detection, 109
every, 185, 187
in replications, 109
locking, 123, 166
not errors, 258, 285
not in Message Agent output, 258, 285
primary key, 152, 192
reporting, 149
resolving, 144, 146, 183
SQL Remote handling of, 143, 183
VERIFY_ALL_COLUMNS option, 145
connections
Message Agent, 240
consolidate permissions, 367, 374, 375
granting, 221
revoking, 221
CONSOLIDATE permissions
granting, 224
consolidated database, 374, 375
publishing, 376
consolidated databases, 13
setting up, 44, 53, 72, 82
consolidated permissions
managing, 219
constraints
the extraction utility, 214
continuous mode
Message Agent, 239
control statements
replication of, 275
conventions
documentation, x
CREATE PUBLICATION statement
syntax, 356
CREATE REMOTE MESSAGE TYPE statement
syntax, 358
CREATE statements
replication of, 104
CREATE SUBSCRIPTION statement
about, 157, 216
syntax, 360
CREATE TRIGGER statement
syntax, 361
creating
message types, 229, 230
publications, 356
subscriptions, 360
creating articles
column-wise partitioning, 162
row-wise partitioning, 163
creating publications, 47, 55, 74, 115, 161
column-wise partitioning, 116, 162
row-wise partitioning, 117, 163
simple publications, 115, 161
using Sybase Central, 115, 161
creating subscriptions, 48, 55, 75, 85, 157, 198
current publisher
table for, 336
CURRENT PUBLISHER, 46, 369
about, 54
current remote user
conflict resolution, 187
table for, 336
CURRENT REMOTE USER default, 146
cursors
pass-through mode, 275
replication and, 275

D
data recovery
SQL Remote, 241
Database extraction utility, 315
databases
- consolidated, 13
- loading data into, 209
- procedures before extracting, 210
- remote, 13, 49, 57, 77
- synchronizing, 57, 58

dbcsettrunc
- using, 288

dbo user
- system objects, 314
- dbremote, 59, 60, 88, 89
- about, 306
- commline, 306
- Message Agent, 88

DBREMOTE
- about, 239
- security, 257, 283

DBUNLOAD
- replication, 272

DBXTRACT, 57
- about, 207
- using, 209

DDL statements
- replication of, 104
- SQL Remote, 289

debug control parameter
- FTP message system, 233

Debug control parameter
- file message system, 232
- MAPI message system, 236
- SMTP message system, 235
- VIM message system, 237

defaults
- the extraction utility, 214

DELETE statement
- replication of, 100

DELETE_OLD_LOGS
- replication option, 323

DELETE_OLD_LOGS option, 266

Deleting Corrupt Message error, 250
deployment
- of SQL Remote databases, 203, 204, 205
design
- about, 95, 96
- at the consolidated database, 96
- for Adaptive Server Anywhere, 113
- for Adaptive Server Enterprise, 159
- for mobile workforces, 119, 164
- locking, 123, 166
- many-to-many example, 133, 137, 175
- many-to-many relationships, 133, 175

of publications, 122, 166
- performance, 124
- tasks, 114, 160
designing publications, 109, 149, 190, 191
directories
- executable, xiii
- installation, xiii

Directory control parameter
- file message system, 232
documentation
- conventions, x

DROP PUBLICATION statement
- about, 120
- syntax, 364

DROP REMOTE MESSAGE TYPE statement
- syntax, 365

DROP SUBSCRIPTION statement
- syntax, 366

dropping
- message types, 230, 231
- publications, 364
- subscriptions, 366

dropping publications, 120
dropping users
- consolidate, 374, 375
dsi_data_style parameter
- configuring, 300
dsi_num_threads
- Replication Server parameter, 300
dumps
- coordinating, 302

E

e-mail, 15
- MAPI, 228
- SMTP, 228
- VIM, 228
encoding
- about, 250
- custom, 251
encryption
- of messages, 242
environment variable
- Message Agent options, 308
environment variables
- SQLREMOTE, 231
error handling, 258, 285
    about, 259, 285
default, 258, 285
ignoring errors, 258
errors
    conflicts are not, 258, 285
    handling, 259, 285
    ignoring, 258
    in SQL statements, 110
    notification, 259, 285
    primary key, 109
    reporting, 258, 285
types of in replication, 109
executable directory
    about, xiii
extraction
    custom procedures, 211
designing a procedure, 211
    of databases, 203, 204, 207
do of many databases, 211
    operating systems, 207
    procedures before, 210
    reload files, 209
    using Sybase Central, 210
extraction utility, 209
    about, 207, 214, 215
    for Adaptive Server Enterprise, 214, 215
    limits, 212
    options, 325
    procedures before using, 210
    purpose, 212
    using from Sybase Central, 210

F
FILE message type
    about, 354
FILE messages
    about, 358, 365
file sharing
    control parameters, 232
    message system, 228
    message types, 232
    operating systems supported, 232
Fire
    triggers option
    trigger actions, 103
Force
    Download control parameter
    MAPI message system, 236
foreign keys
    publications, 122, 125, 166, 168, 171
territory realignment, 128, 129, 170
frequency
    of sending, 223
    of sending messages, 367, 370
ftp
    control parameters, 233
    message types, 233
    operating systems supported, 233
ftp message type, 228
    about, 354
FTP messages
    about, 358

G
GRANT CONSOLIDATE statement, 221
    syntax, 367
GRANT PUBLISH, 45, 53
GRANT PUBLISH statement, 219, 220
    syntax, 369
GRANT REMOTE, 45, 53
GRANT REMOTE DBA statement
    syntax, 372
GRANT REMOTE statement, 221
    syntax, 370
    granting consolidate permissions, 221
    granting publish permissions, 219, 220
    granting remote permissions, 221

H
Hewlett Packard HP-UX
    availability on, 5
host control parameter
    FTP message system, 233

I
IBM AIX
    availability on, 5
IBM OS/2
    availability on, 5
icons
    used in manuals, xi
IF statement
   passthrough mode, 275
   replication of, 275
IF UPDATE clause
   in triggers, 361
IMAGE data type
   replication of, 104
INSERT statement
   replication of, 100
installation directory
   about, xiii
international
   character set support, 97
Internet
   e-mail, 233
   SQL Remote, 233
intranets
   mobile, 6
IPM_Receive control parameter
   MAPI message system, 236
IPM_Send control parameter
   MAPI message system, 236

J
joins
   updates, 382
   updates based on, 383

L
1 option
   Message Agent, 242
laptop computers, 24
   replication, 24
   SQL Remote, 24
limitations
   conflict resolution, 185
   Enterprise to Enterprise, 438
loading databases, 209
locking
   in a replications system, 123, 166
   publication design, 152, 192
log management
   SQL Remote, 241
log transfer interface
   the Message Agent, 288
LONG BINARY
   replication, 104
LONG BINARY data type
   replication of, 104
LONG VARCHAR
   replication, 104
LONG VARCHAR data type
   replication of, 104
LOOP statement
   passthrough mode, 275
Lotus Notes
   SQL Remote, 228, 237
LTM
   SQL Remote, 279

M
m option
   Message Agent, 244
maintenance releases
   upgrading, 205
many-to-many relationships
   example, 133, 136, 175, 176
   publication design, 133, 175
   Subscribe_by_remote option, 140, 180
   territory realignment, 137
   triggers for, 138
MAPI
   control parameters, 235, 236
   operating systems supported, 236
MAPI message type
   about, 354
MAPI messages
   about, 358, 365
MAPI system, 228
marker SQL Remote table
   about, 337
media failure
   SQL Remote, 241
Message Agent, 18, 59, 60, 88, 89
   about, 239, 283
   batch mode, 239
   cmdline, 306
   connections, 240
   continuous mode, 239
   delivering messages, 252, 254
   -l option, 242
   -m option, 244
   message tracking, 252, 254
   output, 258, 285
   performance, 243, 247
   polling, 245
missing messages
about, 246
mobile intranet, 6
mobile workforces, 22, 24
publication design, 119, 164
multi-tier installations
passthrough statements, 275
permissions, 226

N
named constraints
the extraction utility, 214
named defaults
the extraction utility, 214
NCHAR data type
the extraction utility, 214
NetWare
SQL Remote, 232
supported message systems, 228
Notes
SQL Remote, 228, 237
Novell NetWare
availability on, 5
NUMBER function
updates, 383
NVARCHAR data type
the extraction utility, 214

O
object SQL Remote table
about, 338
offsets
in transaction log, 252
operating systems
availability on, 5
supported, 4
option SQL Remote table
about, 338
options
Subscribe_by_remote, 140, 180
the extraction utility, 325
Verify_threshold, 105
partitioning
  column-wise, 116, 162
  row-wise, 117, 163
passthrough mode
  Adaptive Server Enterprise, 290
  batches, 275
  for Adaptive Server Anywhere, 273
  operations not replicated, 275
  operations not replicated in, 275
  starting, 373
  stopping, 373
  uses, 274
passthrough SQL Remote table
  about, 338
PASSTHROUGH statement
  syntax, 373
  using, 273
passthrough statements
  multi-tier installations, 275
password control parameter
  FTP message system, 233
Password control parameter
  VIM message system, 237
passwords
  the extraction utility, 214
Path control parameter
  VIM message system, 237
patience
  Message Agent, 246
performance
  design tips, 124
  incoming messages, 245
  Message Agent, 243
  message sending, 247
  number of subscriptions, 108
  publications, 107
  replication throughput, 243
  replication turnaround time, 243
SQL Remote, 243
  threading, 243
  updates, 384
permissions
  consolidate, 221
  CONSOLIDATE, 367
  granting CONSOLIDATE, 224
  multi-tier installations, 226
  publish, 45, 53, 219, 220
  remote, 45, 53, 221
  remote dba, 372, 378
  revoking CONSOLIDATE, 374, 375
  revoking PUBLISH, 376
  revoking REMOTE, 225
SQL Remote administration, 218
platforms
  supported operating systems, 4
polling
  messages, 245
pop3_host control parameter
  SMTP message system, 235
pop3_password control parameter
  SMTP message system, 235
pop3_userid control parameter
  SMTP message system, 235
port control parameter
  FTP message system, 233
PowerDynamo
  SQL Remote, 6
primary key errors
  about, 109
primary key pool
  about, 152, 192
primary key pools
  about, 192
  replenishing, 154, 194
  replicating, 153, 193
  summary, 156, 197
primary keys
  in publications, 100
  publications, 122, 166
  replication, 149, 191
SQL Remote, 149, 191
  uniqueness, 152, 192
procedure groups
  the extraction utility, 214
procedures
  passthrough mode, 275
  replicating, 102
SQL Remote, 102
SQL Remote Open Server, 302
  statements corresponding to, 437
profiles
  Microsoft Exchange, 236
publication design
  for Adaptive Server Anywhere, 113
  for Adaptive Server Enterprise, 159
  for many subscribers, 119, 164
  using subscription expression, 119, 164
publication SQL Remote table
  about, 328, 339
publications
about, 20
column-wise partitioning, 116, 162
creating, 47, 55, 74, 115, 161, 356
creating using Sybase Central, 115, 161
design at consolidated database, 114
designing, 95, 96, 109, 122, 149, 166, 190, 191, 384
dropping, 120, 364
element, 63
foreign keys, 122, 125, 166, 168
locking, 123, 166
many-to-many example, 133, 136, 137, 175, 176
many-to-many relationships, 133, 175
notes, 120
performance, 107, 124
primary keys, 122, 149, 152, 166, 191, 192
referential integrity, 149, 190
row-wise partitioning, 117, 163
setting up, 115, 161
simple, 115, 161
subqueries, 125
tables in many publications, 107
transactions, 123, 166
updates, 384
using a WHERE clause, 118, 163
whole tables, 117
publications SQL Remote view
about, 332, 343
publish permissions, 45, 53, 369
granting, 219, 220
managing, 219
remote permissions, 45, 53
revoking, 219, 220, 376
publisher, 46
about, 219, 220
adding to a database, 54
address, 229, 354, 358, 365
creating, 219, 220
GRANT PUBLISH statement, 369
remote, 370
publisher SQL Remote table
about, 339

Q
Qualify owners
replication option, 324
quiet system
definition, 289

R
Receive_all control parameter
VIM message system, 237
receiving messages, 60, 89
recovery
SQL Remote, 241
referential integrity
replication, 149, 190
SQL Remote, 149, 190
registry
SQL Remote, 231
reload files
database extraction, 209
remote databases, 13
remote permissions, 222
setting up, 49, 57, 58, 77
REMOTE DBA permissions
granting, 372
revoking, 378
remote permissions, 370
granting, 221
managing, 219
revoking, 221, 225, 377
Sybase Central, 222
remote users
REVOKE REMOTE statement, 377
remotetable SQL Remote table
about, 339
remotetables SQL Remote view
about, 344
remotetype SQL Remote table
about, 329, 340
remotetypes SQL Remote view
about, 344
remoteuser SQL Remote table
about, 329, 340
remoteuser table, 252
remoteusers SQL Remote view
about, 333, 344
replicating procedures, 102
replicating triggers, 103
replication
administering, 22
backup procedures, 262, 266, 271, 286
by e-mail, 15
case studies, 24, 26
conflicts, 109
data types, 104
design, 122, 166
designing, 122, 166
message-based, 15
mobile workforces, 24
of blobs, 104
of control statements, 275
of cursor operations, 275
of cursor statements, 275
of data definition statements, 104
of procedures, 102
of SQL statements, 273
of stored procedures, 275
of triggers, 103
options, 323
passthrough mode, 273
primary key errors, 149, 191
primary keys, 152, 192
publications, 20
referential integrity errors, 149, 190
server-to-laptop replication, 24
server-to-server, 26
setup examples, 24
subscriptions, 20
transaction log and, 22, 100
transaction log management, 262, 266, 271, 286
triggers, 150
upgrading databases, 271
Replication, 315
dbremote, 306
Message Agent, 306
ssqueue, 321
ssremote, 306
synchronization, 315
Replication Agent
SQL Remote, 279, 292, 293
replication conflicts
about, 142, 182
replication definitions
Replication Server, 300
replication options
Qualify_owners, 324
Quote_all_identifiers, 324
REPLICATION_ERROR, 110, 259, 285, 324
SUBSCRIBE_BY_REMOTE, 325
VERIFY_THRESHOLD, 325
replication role
granting, 283
Replication Server
configuring, 299, 300
restart the connection, 301
SQL Remote, 279, 291, 292, 299, 300
SQL Remote architecture, 293
ssqueue, 296
using SQL Remote with, 321
REPLICATION_ERROR
replication option, 110, 259, 285, 324
reporting
conflicts, 149
reporting errors, 258, 285
resend requests
about, 246
messages, 245
RESOLVE UPDATE triggers, 144, 146
REVOKE CONSOLIDATE statement, 221
syntax, 374, 375
REVOKE PUBLISH statement, 219, 220
syntax, 376
REVOKE REMOTE DBA statement
syntax, 378
REVOKE REMOTE statement, 221, 225
syntax, 377
revoking consolidate permissions, 221
revoking publish permissions, 219, 220
revoking remote permissions, 221, 225
roles
the extraction utility, 214
root control parameter
FTP message system, 233
rs_dumpdb
using, 302
rs_dumptran
using, 302
running
the Message Agent, 256

S
sample database
about, xiv
schema changes
SQL Remote, 289
SQL Remote Open Server, 302
security
replication, 372, 378
ssxtract, 314
SEND AT
frequency setting, 223

449
SEND AT clause, 367, 370
publish, 369
SEND EVERY
frequency setting, 223
SEND EVERY clause, 367, 370
send frequency
selecting, 223
the Message Agent, 239
Send VIM Mail control parameter
VIM message system, 237
sending messages, 59, 88, 89
server-to-server replication, 26
services
Message Agent as, 256
setting up a remote database, 49, 57, 77
setting up publications, 115, 161
setting up subscriptions, 157, 198
setting up the consolidated database, 44, 53, 72, 82
Silicon Graphics IRIX
availability on, 5
SMTP
control parameters, 234
e-mail, 233
operating systems supported, 233
SQL Remote, 233
SMTP message type, 228
SMTP/POP
addresses, 234
smtp_host control parameter
SMTP message system, 235
software
dbremote, 306
ssqueue, 321
ssremote, 306
SSXTRACT, 315
sp_procedure
syntax, 409
sp_add_article procedure
syntax, 387
sp_add_article_col procedure
syntax, 388
sp_add_remote_table procedure
syntax, 389
sp_create_publication procedure
syntax, 391
sp_drop_publication procedure
syntax, 392
sp_drop_remote_type procedure
syntax, 393
sp_drop_sql remote procedure
syntax, 394
sp_grant_consolidate procedure
syntax, 395
sp_grant_remote procedure
syntax, 397
sp_modify_article procedure
syntax, 399
sp_modify_remote_table procedure
syntax, 401
sp_passthrough procedure
about, 290
syntax, 403
sp_passthrough_piece procedure
about, 290
syntax, 404
sp_passthrough_stop procedure
about, 290
syntax, 406
sp_passthrough_subscription procedure
about, 290
syntax, 407
sp_passthrough_user procedure
about, 290
syntax, 408
sp_populate_sql_anywhere procedure
about, 215
sp_publisher procedure
syntax, 410
sp_queue_clean procedure
syntax, 411
sp_queue_delete_old procedure
syntax, 412
sp_queue_drop procedure
syntax, 413
sp_queue_dump_database procedure
syntax, 414
sp_queue_dump_transaction procedure
syntax, 415
sp_queue_get_state procedure
syntax, 416
sp_queue_read procedure
syntax, 417
sp_queue_reset procedure
syntax, 418
sp_queue_set_confirm procedure
syntax, 419
sp_queue_set_progress procedure
syntax, 420
sp_queue_transaction procedure
syntax, 421
sp_remote procedure
syntax, 422
sp_remote_option procedure
  syntax, 423
sp_remote_type procedure
  syntax, 424
sp_remove_article procedure
  syntax, 425
sp_remove_article_col procedure
  syntax, 426
sp_remove_remote_table procedure
  syntax, 427
sp_revoke_consolidate procedure
  syntax, 428
sp_revoke_remote procedure
  syntax, 429
sp_setreplicate procedure
  sp_add_remote_table, 389
sp_subscription procedure
  about, 198
  syntax, 430
  using, 216
sp_user_extraction_hook
  example, 179
sp_user_extraction_hook procedure
  about, 215
SQL Remote
  about, 9
  administering, 22, 274
  ALTER REMOTE MESSAGE TYPE statement, 354
  altering publications, 353
  articles, 328, 337
  backup procedures, 262, 266, 271, 286
  case studies, 24, 26
  concepts, 13
  consolidate permissions, 367, 374, 375
  consolidated databases, 13
  CREATE REMOTE MESSAGE TYPE statement, 358
  creating publications, 47, 55, 74, 356
  creating subscriptions, 360
dbremote, 59, 60, 88, 89
  DROP REMOTE MESSAGE TYPE statement, 365
  dropping publications, 364
  dropping subscriptions, 366
  extracting a database, 315
  GRANT PUBLISH statement, 369
  granting publish permissions, 45, 53
  granting remote permissions, 45, 53
  Message Agent, 18, 59, 60, 88, 89
  message delivery, 252, 254
message tracking, 252, 254
mobile workforces, 22, 24
multi-tier installations, 275
options, 323
passthrough mode, 373
publications, 20
publish permissions, 376
remote databases, 13
remote permissions, 370, 377
server-to-laptop replication, 24
server-to-server replication, 26
setting up, 41, 51, 69, 79
setting up a consolidated database, 44, 53, 72, 82
setting up a remote database, 49, 57, 77
setup examples, 24
ssremote, 89
SSXTRACT utility, 315
starting subscriptions, 379
stopping subscriptions, 380
subscribers, 22
subscriptions, 20
synchronization, 315
synchronizing subscriptions, 381
system tables, 328, 337
transaction log management, 262, 266, 271, 286
uninstalling, 394, 413
uploading databases, 272
upgrading, 36
upgrading databases, 271
SQL Remote Open Server
  architecture, 293
  commline, 321
  on IRIX, 5
  procedures, 302
  schema changes, 289
  setting up, 296
  when needed, 292
SQL Remote procedures
  sp_, 409
  sp_add_article, 387
  sp_add_article_col, 388
  sp_add_remote_table, 389
  sp_create_publication, 391
  sp_drop_publication, 392
  sp_drop_remote_type, 393
  sp_drop_sql_remote, 394
  sp_grant_consolidate, 395
  sp_grant_remote, 397
  sp_modify_article, 399
  sp_modify_remote_table, 401
  sp_passthrough, 403
sp_passsthrough_piece, 404
sp_passsthrough_stop, 406
sp_passsthrough_subscription, 407
sp_passsthrough_user, 408
sp_publisher, 410
sp_queue_clean, 411
sp_queue_delete_old, 412
sp_queue_drop, 413
sp_queue_dump_database, 414
sp_queue_dump_transaction, 415
sp_queue_get_state, 416
sp_queue_read, 417
sp_queue_reset, 418
sp_queue_set_confirm, 419
sp_queue_set_progress, 420
sp_queue_transaction, 421
sp_remote, 422
sp_remote_option, 423
sp_remote_type, 424
sp_remove_article, 425
sp_remove_article_col, 426
sp_remove_remote_table, 427
sp_revolve_consolidate, 428
sp_revolve_remote, 429
sp_subscription, 430
SQL Remote system tables
article, 328, 336
articlecol, 328, 337
marker, 337
object, 338
option, 338
passsthrough, 338
publication, 328, 339
publisher, 339
remotetable, 339
remotetype, 329, 340
remoteuser, 329, 340
subscription, 331, 341
SQL Remote system views
articlecols, 332, 343
articles, 332, 343
publications, 332, 343
remotetables, 344
remotetypes, 344
remoteusers, 333, 344
subscriptions, 334, 346
SQL Remote tables
#remote, 336
SQL statements
replication of, 100, 101, 104
SQLANY.INI
SQL Remote, 231
SQLRemote environment variable, 231
SQL_REMOTE environment variable
alternative to, 232
squpdate.sql
upgrading the stable queue, 36
sr_article table
about, 336
sr_articlecol table
about, 337
sr_marker table
about, 337
sr_object table
about, 338
sr_option table
about, 338
sr_passsthrough table
about, 338
sr_publication table
about, 339
sr_publisher table
about, 339
sr_queue_state table
about, 347
sr_remotetable table
about, 339
sr_remotetype table
about, 340
sr_remoteuser table
about, 340
sr_subscription table
about, 341
sr_transaction table
about, 348
ssqueue
about, 291, 321
architecture, 293
commline, 321
on IRIX, 5
setting up, 296
when needed, 292
ssremote, 89
about, 306
commline, 306
SSREMOTE
about, 239, 283
ssupdate.sql
upgrading SQL Remote, 36
ssxtract
about, 207
using, 209
SSXTRACT
about, 315
command line, 315
using, 214, 215
stable queue
cleaning, 309
Replication Server, 294
SQL Remote Open Server, 294
ssqueue, 294
system tables, 347
START SUBSCRIPTION statement
syntax, 379
starting
the Message Agent, 256
starting subscriptions, 379
statements
replication of, 100, 101
STOP SUBSCRIPTION statement
syntax, 380
stopping subscriptions, 380
store-and-forward, 15
stored procedures
passthrough mode, 275
replication of, 102
SQL Remote Open Server, 302
subqueries
in SQL Remote, 125
publications, 125
SUBSCRIBE BY clause, 356
SUBSCRIBE_BY_REMOTE
replication option, 325
Subscribe_by_remote option
many-to-many relationships, 140, 180
subscription expressions
about, 119, 164
cost of evaluating, 107
in the transaction log, 107
many-valued, 107
subqueries, 125
subscription SQL Remote table
about, 331, 341
subscription-list columns
about, 171
in SQL Remote, 168
maintaining, 173, 177, 179
triggers, 177
triggers for, 173
subscriptions
about, 20
creating, 48, 55, 75, 85, 157, 198, 360
dropping, 366
performance, 108
setting up, 157, 198
starting, 379
stopping, 380
synchronizing, 209, 216, 381
the Message Agent, 108
updates, 384
subscriptions SQL Remote view
about, 334, 346
Sun Solaris
availability on, 5
Sybase Central
creating publications with, 115, 161
extraction utility, 210
message types, 229, 230
synchronization
about, 207
do databases, 203, 204
operating systems, 207
using DBXTRACT, 209
using ssxtract, 209
using the extraction utility, 209
synchronization utility, 57
SYNCHRONIZE SUBSCRIPTION statement
about, 216
syntax, 381
synchronizing databases, 57, 58
using the message system, 216
synchronizing subscriptions, 381
SYSREMOTEUSER table, 252
system objects
the dbo user, 314
system tables
stable queue, 347
SYSARTICLE, 328, 337
tables
column-wise partitioning, 116, 162
in publications, 117
publishing, 115, 161
row-wise partitioning, 117, 163
territory realignment
about, 173
many-to-many relationships, 137
replication of UPDATES, 101
subscription-list columns, 171
triggers for, 138
territory realignments
foreign keys, 128, 129, 170
testing
SQL Remote, 205
TEXT data type
replication of, 104
threading
Message Agent, 243
timestamp columns
the extraction utility, 214
transaction log
Adaptive Server Enterprise, 288
managing, 288
Message Agent, 306
message tracking, 252
offsets, 252
publications, 107
replication, 22, 100
scanning, 292
SQL Remote, 22, 100, 306
update statements, 107
transaction log mirror
replication, 262, 286
transactions
replication, 100
triggers
conflict resolution, 144, 146, 183
replicating, 103
replication, 130, 150
replication option, 103
RESOLVE UPDATE, 144, 146
SQL Remote, 103, 150
subscription-list columns, 173, 177
territory realignment, 129, 130
Watcom-SQL, 361
truncation point
in transaction log, 288
setting, 288
turnaround time
replication performance, 243
uninstalling
from a database, 394, 413
UNIX
supported message systems, 228
unloading databases
replication, 272
UPDATE
IF UPDATE clause, 361
UPDATE conflicts
about, 142, 182
UPDATE statement
conflict detection, 105
for publications, 129
replication of, 100, 101, 128, 129, 170
syntax, 382
territory realignment, 101
UPDATE statements
conflict detection, 101
updates
information in transaction log, 107
joins, 382, 383
upgrading
about, 36
replication, 271
SQL Remote, 271
SQL Remote installations, 205
user control parameter
FTP message system, 233
Userid control parameter
VIM message system, 237
users
remote, 13

V
VERIFY_ALL_COLUMNS
replication option, 325
VERIFY_ALL_COLUMNS option, 145
VERIFY_THRESHOLD
replication option, 325
Verify_threshold option
message size, 105
VIM
control parameters, 237
message system, 237
operating systems supported, 237
VIM message type, 228
about, 354
VIM messages
about, 358, 365
W

WHERE clause
  in publications, 118, 163
Windows 3.x
  availability on, 5
  supported message systems, 228

Windows 95
  availability on, 5
  supported message systems, 228
Worker threads
  Message Agent, 243