



SQL Remote™ User's Guide

Part number: 38133-01-0900-01

Last modified: June 2003

Copyright © 1989–2003 Sybase, Inc. Portions copyright © 2001–2003 iAnywhere Solutions, Inc. All rights reserved.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of iAnywhere Solutions, Inc. iAnywhere Solutions, Inc. is a subsidiary of Sybase, Inc.

Sybase, SYBASE (logo), AccelaTrade, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, ASEP, AvantGo, AvantGo Application Alerts, AvantGo Mobile Delivery, AvantGo Mobile Document Viewer, AvantGo Mobile Inspection, AvantGo Mobile Marketing Channel, AvantGo Mobile Pharma, AvantGo Mobile Sales, AvantGo Pylon, AvantGo Pylon Application Server, AvantGo Pylon Conduit, AvantGo Pylon PIM Server, AvantGo Pylon Pro, Backup Server, BayCam, Bit-Wise, BizTracker, Certified PowerBuilder Developer, Certified SYBASE Professional, Certified SYBASE Professional (logo), ClearConnect, Client Services, Client-Library, CodeBank, Column Design, ComponentPack, Connection Manager, Convoy/DM, Copernicus, CSP, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, Dynamic Mobility Model, Dynamo, e-ADK, E-Anywhere, e-Biz Integrator, E-Whatever, EC Gateway, ECMAP, ECRT, eFulfillment Accelerator, Electronic Case Management, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise Portal (logo), Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, eremote, Everything Works Better When Everything Works Together, EWA, Financial Fusion, Financial Fusion (and design), Financial Fusion Server, Formula One, Fusion Powered e-Finance, Fusion Powered Financial Destinations, Fusion Powered STP, Gateway Manager, GeoPoint, GlobalFIX, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InstaHelp, InternetBuilder, iremote, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Logical Memory Manager, M-Business Channel, M-Business Network, M-Business Server, Mail Anywhere Studio, MainframeConnect, Maintenance Express, Manage Anywhere Studio, MAP, MDI Access Server, MDI Database Gateway, media.splash, Message Anywhere Server, MetaWorks, MethodSet, ML Query, MobicATS, My AvantGo, My AvantGo Media Channel, My AvantGo Mobile Marketing, MySupport, Net-Gateway, Net-Library, New Era of Networks, Next Generation Learning, Next Generation Learning Studio, O DEVICE, OASIS, OASIS (logo), ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Biz, Open Business Interchange, Open Client, Open Client/Server, Open Client/Server Interfaces, Open ClientConnect, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, Partnerships that Work, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, PhysicalArchitect, Pocket PowerBuilder, PocketBuilder, Power Through Knowledge, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, Powering the New Economy, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, Powersoft Portfolio, Powersoft Professional, PowerStage, PowerStudio, PowerTips, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, QAnywhere, Rapport, Relational Beans, RepConnector, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Report Workbench, Report-Execute, Resource Manager, RW-DisplayLib, RW-Library, S.W.I.F.T. Message Format Libraries, SAFE, SAFE/PRO, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL Server SNMP SubAgent, SQL Server/CFT, SQL Server/DBM, SQL SMART, SQL Station, SQL Toolset, SQLJ, Stage III Engineering, Startup.Com, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Development Framework, Sybase Financial Server, Sybase Gateways, Sybase Learning Connection, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase User Workbench, Sybase Virtual Server Architecture, SybaseWare, Syber Financial, SyberAssist, SybMD, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, The Enterprise Client/Server Company, The Extensible Software Platform, The Future Is Wide Open, The Learning Connection, The Model For Client/Server Solutions, The Online Information Center, The Power of One, TradeForce, Transact-SQL, Translation Toolkit, Turning Imagination Into Reality, UltraLite, UltraLite.NET, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Versacore, Viewer, VisualWriter, VQL, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, WarehouseArchitect, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server, and XP Server are trademarks of Sybase, Inc. or its subsidiaries.

Certicom, MobileTrust, and SSL Plus are trademarks and Security Builder is a registered trademark of Certicom Corp. Copyright © 1997–2001 Certicom Corp. Portions are Copyright © 1997–1998, Consensus Development Corporation, a wholly owned subsidiary of Certicom Corp. All rights reserved. Contains an implementation of NR signatures, licensed under U.S. patent 5,600,725. Protected by U.S. patents 5,787,028; 4,745,568; 5,761,305. Patents pending.

All other trademarks are property of their respective owners.

Contents

About This Manual	ix
SQL Anywhere Studio documentation	x
Documentation conventions	xiii
The Adaptive Server Anywhere sample database	xv
Finding out more and providing feedback	xvi
I Introduction to SQL Remote	1
1 Welcome to SQL Remote	3
About SQL Remote	4
About this manual	5
2 SQL Remote Concepts	7
SQL Remote components	8
Publications and subscriptions	11
SQL Remote features	13
Some sample installations	15
3 Setting Up SQL Remote	19
Setup overview	20
Preparing your Adaptive Server Enterprise server	21
Upgrading SQL Remote for Adaptive Server Enterprise	25
Uninstalling SQL Remote	26
4 Tutorials for Adaptive Server Anywhere Users	27
Introduction	28
Tutorial: Adaptive Server Anywhere replication using Sybase Central	32
Tutorial: Adaptive Server Anywhere replication using Interactive SQL	
and dbxtract	40
Start replicating data	47
A sample publication	51
5 A Tutorial for Adaptive Server Enterprise Users	53
Introduction	54
Tutorial: Adaptive Server Enterprise replication	57
Start replicating data	66

II	Replication Design for SQL Remote	71
6	Principles of SQL Remote Design	73
	Design overview	74
	How statements are replicated	78
	How data types are replicated	83
	Who gets what?	86
	Replication errors and conflicts	88
7	SQL Remote Design for Adaptive Server Anywhere	91
	Design overview	92
	Publishing data	93
	Publication design for Adaptive Server Anywhere	102
	Partitioning tables that do not contain the subscription expression	105
	Sharing rows among several subscriptions	112
	Managing conflicts	120
	Ensuring unique primary keys	129
	Creating subscriptions	139
8	SQL Remote Design for Adaptive Server Enterprise	141
	Design overview	142
	Creating publications	143
	Publication design for Adaptive Server Enterprise	147
	Partitioning tables that do not contain the subscription column	149
	Sharing rows among several subscriptions	157
	Managing conflicts	165
	Ensuring unique primary keys	175
	Creating subscriptions	181
III	SQL Remote Administration	183
9	Deploying and Synchronizing Databases	185
	Deployment overview	186
	Test before deployment	187
	Synchronizing databases	189
	Using the extraction utility	191
	Synchronizing data over a message system	198
10	SQL Remote Administration	199
	Management overview	200
	Managing SQL Remote permissions	201
	Using message types	210
	Running the Message Agent	223

Tuning Message Agent performance	228
Encoding and compressing messages	235
The message tracking system	237
11 Administering SQL Remote for Adaptive Server Anywhere	241
Running the Message Agent	242
Error reporting and handling	245
Transaction log and backup management	249
Using passthrough mode	260
12 Administering SQL Remote for Adaptive Server Enterprise	263
How the Message Agent for Adaptive Server Enterprise works	264
Running the Message Agent	269
Error reporting and handling	271
Adaptive Server Enterprise transaction log and backup management	272
Making schema changes	275
Using passthrough mode	276
13 Using SQL Remote with Replication Server	277
When you need to use the SQL Remote Open Server	278
Architecture for Replication Server/SQL Remote installations	279
Setting up SQL Remote Open Server	282
Configuring Replication Server	285
Other issues	287
IV Reference	289
14 Utilities and Options Reference	291
The Message Agent	292
The Database Extraction utility	302
The SQL Remote Open Server	310
SQL Remote options	313
SQL Remote event-hook procedures	318
15 System Objects for Adaptive Server Anywhere	323
SQL Remote system tables	324
SQL Remote system views	331
16 System Objects for Adaptive Server Enterprise	335
SQL Remote system tables	336
SQL Remote system views	344
Stable Queue tables	348

17 Command Reference for Adaptive Server Anywhere	351
ALTER REMOTE MESSAGE TYPE statement	353
CREATE PUBLICATION statement	354
CREATE REMOTE MESSAGE TYPE statement	355
CREATE SUBSCRIPTION statement	356
CREATE TRIGGER statement	357
DROP PUBLICATION statement	359
DROP REMOTE MESSAGE TYPE statement	360
DROP SUBSCRIPTION statement	361
GRANT CONSOLIDATE statement	362
GRANT PUBLISH statement	363
GRANT REMOTE statement	364
GRANT REMOTE DBA statement	365
PASSTHROUGH statement	366
REMOTE RESET statement	367
REVOKE CONSOLIDATE statement	368
REVOKE PUBLISH statement	369
REVOKE REMOTE statement	370
REVOKE REMOTE DBA statement	371
SET REMOTE OPTION statement	372
START SUBSCRIPTION statement	373
STOP SUBSCRIPTION statement	374
SYNCHRONIZE SUBSCRIPTION statement	375
UPDATE statement	376
 18 Command Reference for Adaptive Server Enterprise	 377
sp_add_article procedure	379
sp_add_article_col procedure	381
sp_add_remote_table procedure	382
sp_create_publication procedure	384
sp_drop_publication procedure	385
sp_drop_remote_type procedure	386
sp_drop_sql_remote procedure	387
sp_grant_consolidate procedure	388
sp_grant_remote procedure	391
sp_link_option procedure	394
sp_modify_article procedure	396
sp_modify_remote_table procedure	398
sp_passthrough procedure	400
sp_passthrough_piece procedure	401
sp_passthrough_stop procedure	403
sp_passthrough_subscription procedure	404
sp_passthrough_user procedure	405

sp_populate_sql_anywhere procedure	406
sp_publisher procedure	407
sp_queue_clean procedure	408
sp_queue_confirmed_delete_old procedure	409
sp_queue_confirmed_transaction procedure	410
sp_queue_delete_old procedure	411
sp_queue_drop procedure	412
sp_queue_dump_database procedure	413
sp_queue_dump_transaction procedure	414
sp_queue_get_state procedure	415
sp_queue_log_transfer_reset 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	425
sp_remove_article procedure	426
sp_remove_article_col procedure	427
sp_remove_remote_table procedure	428
sp_revoke_consolidate procedure	429
sp_revoke_remote procedure	430
sp_subscription procedure	431
sp_subscription_reset procedure	432

V Appendix 433

A SQL Remote for Adaptive Server Enterprise and Adaptive Server Any-where: Differences 435


Types of difference	436
Differences in functionality	437
Differences in approach	438
Limitations for Enterprise to Enterprise replication	440

B Supported Platforms and Message Links 443

Supported message systems	444
Supported operating systems	445

Index 447

About This Manual

Subject	This book describes all aspects of the SQL Remote data replication system for mobile computing, which enables sharing of data between a single Adaptive Server Anywhere or Adaptive Server Enterprise database and many Adaptive Server Anywhere databases using an indirect link such as e-mail or file transfer.
Audience	This book is for users of Adaptive Server Anywhere and Adaptive Server Enterprise who wish to add SQL Remote replication to their information systems.
Before you begin	 For a comparison of SQL Remote with other replication technologies, see “Replication Technologies” [<i>Introducing SQL Anywhere Studio</i> , page 19].

SQL Anywhere Studio documentation

The SQL Anywhere Studio documentation

This book is part of the SQL Anywhere documentation set. This section describes the books in the documentation set and how you can use them.

The SQL Anywhere Studio documentation is available in a variety of forms: in an online form that combines all books in one large help file; as separate PDF files for each book; and as printed books that you can purchase. The documentation consists of the following books:

- ◆ **Introducing SQL Anywhere Studio** This book provides an overview of the SQL Anywhere Studio database management and synchronization technologies. It includes tutorials to introduce you to each of the pieces that make up SQL Anywhere Studio.
- ◆ **What's New in SQL Anywhere Studio** This book is for users of previous versions of the software. It lists new features in this and previous releases of the product and describes upgrade procedures.
- ◆ **Adaptive Server Anywhere Getting Started** This book is for people new to relational databases or new to Adaptive Server Anywhere. It provides a quick start to using the Adaptive Server Anywhere database-management system and introductory material on designing, building, and working with databases.
- ◆ **Adaptive Server Anywhere Database Administration Guide** This book covers material related to running, managing, and configuring databases and database servers.
- ◆ **Adaptive Server Anywhere SQL User's Guide** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- ◆ **Adaptive Server Anywhere SQL Reference Manual** This book provides a complete reference for the SQL language used by Adaptive Server Anywhere. It also describes the Adaptive Server Anywhere system tables and procedures.
- ◆ **Adaptive Server Anywhere Programming Guide** This book describes how to build and deploy database applications using the C, C++, and Java programming languages. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools. It also describes the Adaptive Server Anywhere ADO.NET data provider.

- ◆ **Adaptive Server Anywhere Error Messages** This book provides a complete listing of Adaptive Server Anywhere error messages together with diagnostic information.
- ◆ **SQL Anywhere Studio Security Guide** This book provides information about security features in Adaptive Server Anywhere databases. Adaptive Server Anywhere 7.0 was awarded a TCSEC (Trusted Computer System Evaluation Criteria) C2 security rating from the U.S. Government. This book may be of interest to those who wish to run the current version of Adaptive Server Anywhere in a manner equivalent to the C2-certified environment.
- ◆ **MobiLink Synchronization User's Guide** This book describes how to use the MobiLink data synchronization system for mobile computing, which enables sharing of data between a single Oracle, Sybase, Microsoft or IBM database and many Adaptive Server Anywhere or UltraLite databases.
- ◆ **MobiLink Synchronization Reference** This book is a reference guide to MobiLink command line options, synchronization scripts, SQL statements, stored procedures, utilities, system tables, and error messages.
- ◆ **iAnywhere Solutions ODBC Drivers** This book describes how to set up ODBC drivers to access consolidated databases other than Adaptive Server Anywhere from the MobiLink synchronization server and from Adaptive Server Anywhere remote data access.
- ◆ **SQL Remote User's Guide** This book describes all aspects of the SQL Remote data replication system for mobile computing, which enables sharing of data between a single Adaptive Server Anywhere or Adaptive Server Enterprise database and many Adaptive Server Anywhere databases using an indirect link such as e-mail or file transfer.
- ◆ **SQL Anywhere Studio Help** This book includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools. It is not included in the printed documentation set.
- ◆ **UltraLite Database User's Guide** This book is intended for all UltraLite developers. It introduces the UltraLite database system and provides information common to all UltraLite programming interfaces.
- ◆ **UltraLite Interface Guides** A separate book is provided for each UltraLite programming interface. Some of these interfaces are provided as UltraLite components for rapid application development, and others are provided as static interfaces for C, C++, and Java development.

In addition to this documentation set, PowerDesigner and InfoMaker include their own online documentation.

Documentation formats SQL Anywhere Studio provides documentation in the following formats:

- ◆ **Online documentation** The online documentation contains the complete SQL Anywhere Studio documentation, including both the books and the context-sensitive help for SQL Anywhere tools. The online documentation is updated with each maintenance release of the product, and is the most complete and up-to-date source of documentation.

To access the online documentation on Windows operating systems, choose Start ► Programs ► SQL Anywhere 9 ► Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

To access the online documentation on UNIX operating systems, see the HTML documentation under your SQL Anywhere installation.

- ◆ **Printable books** The SQL Anywhere books are provided as a set of PDF files, viewable with Adobe Acrobat Reader.

The PDF files are available on the CD ROM in the *pdf_docs* directory. You can choose to install them when running the setup program.

- ◆ **Printed books** The complete set of books is available from Sybase sales or from eShop, the Sybase online store. You can access eShop by clicking How to Buy ► eShop at <http://www.ianywhere.com>.

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 appear in upper case, like the words ALTER TABLE in the following example:

ALTER TABLE [*owner*.]*table-name*

- ◆ **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown like the words *owner* and *table-name* in the following example:

ALTER TABLE [*owner*.]*table-name*

- ◆ **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots), like *column-constraint* in the following example:

ADD *column-definition* [*column-constraint*, ...]

One or more list elements are allowed. In this example, if more than one is specified, they must be separated by commas.

- ◆ **Optional portions** Optional portions of a statement are enclosed by square brackets.

RELEASE SAVEPOINT [*savepoint-name*]

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

- ◆ **Options** When none or only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in square brackets.

[**ASC** | **DESC**]

For example, 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 and a bar is used to separate the options.

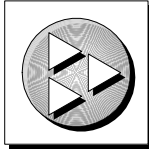
[**QUOTES** { **ON** | **OFF** }]

If the QUOTES option is used, one of ON or OFF must be provided. The brackets and braces should not be typed.

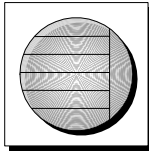
Graphic icons

The following icons are used in this documentation.

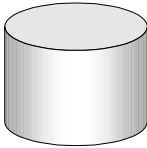
- ◆ A client application.



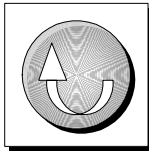
- ◆ A database server, such as Sybase Adaptive Server Anywhere.



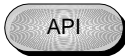
- ◆ A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



- ◆ Replication or synchronization middleware. These assist in sharing data among databases. Examples are the MobiLink Synchronization Server and the SQL Remote Message Agent.



- ◆ A programming interface.



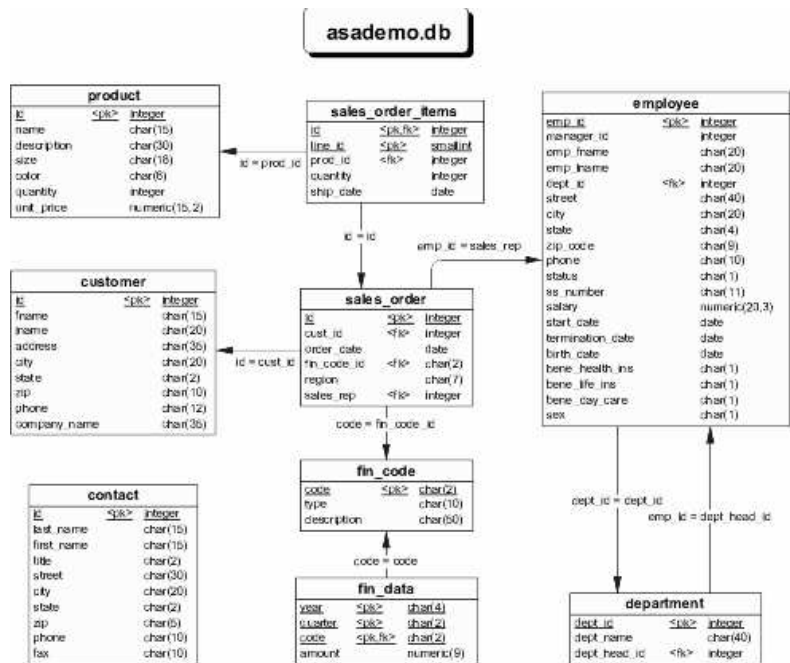
The Adaptive Server Anywhere sample database

Many of the examples throughout the documentation use the Adaptive Server Anywhere sample database.

The sample database is held in a file named *asademo.db*, and is located in your SQL Anywhere directory.

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

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



Finding out more and providing feedback

We would like to receive your opinions, suggestions, and feedback on this documentation.

You can provide feedback on this documentation and on the software through newsgroups set up to discuss SQL Anywhere technologies. These newsgroups can be found on the *forums.sybase.com* news server.

The newsgroups include the following:

- ◆ sybase.public.sqlanywhere.general.
- ◆ sybase.public.sqlanywhere.linux.
- ◆ sybase.public.sqlanywhere.mobilink.
- ◆ sybase.public.sqlanywhere.product_futures_discussion.
- ◆ sybase.public.sqlanywhere.replication.
- ◆ sybase.public.sqlanywhere.ultralite.

Newsgroup disclaimer

iAnywhere Solutions has no obligation to provide solutions, information or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and insure its operation and availability.

iAnywhere Solutions Technical Advisors as well as other staff assist on the newsgroup service when they have time available. They offer their help on a volunteer basis and may not be available on a regular basis to provide solutions and information. Their ability to help is based on their workload.

PART I

INTRODUCTION TO SQL REMOTE

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

Topic:	page
About SQL Remote	4
About this manual	5

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. The time lag between the consolidated and remote databases is configurable, and can range from minutes to 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 Adaptive Server Anywhere 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.

☞ For a detailed introduction to SQL Remote concepts and features, see [“SQL Remote Concepts” on page 7](#).

☞ For a list of supported operating systems and message links, see [“Supported Platforms and Message Links” on page 443](#).

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.

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 (Windows)

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 (UNIX)

1. Consult the instructions for your operating system in the *Adaptive Server Anywhere Read Me First* booklet.

☞ 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 19](#).

CHAPTER 2

SQL Remote Concepts

About this chapter

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

Contents

Topic:	page
SQL Remote components	8
Publications and subscriptions	11
SQL Remote features	13
Some sample installations	15

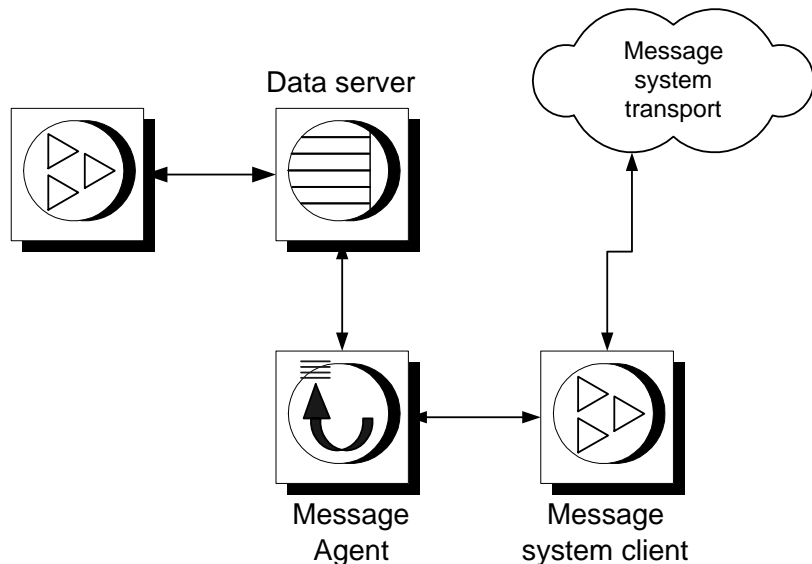
SQL Remote components

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** SQL Remote uses existing message systems to transport replication messages. A file-sharing “message system” is provided, which does not require client software. Each computer involved in SQL Remote replication using a message system other than file sharing must have that message system installed.
- ◆ **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.

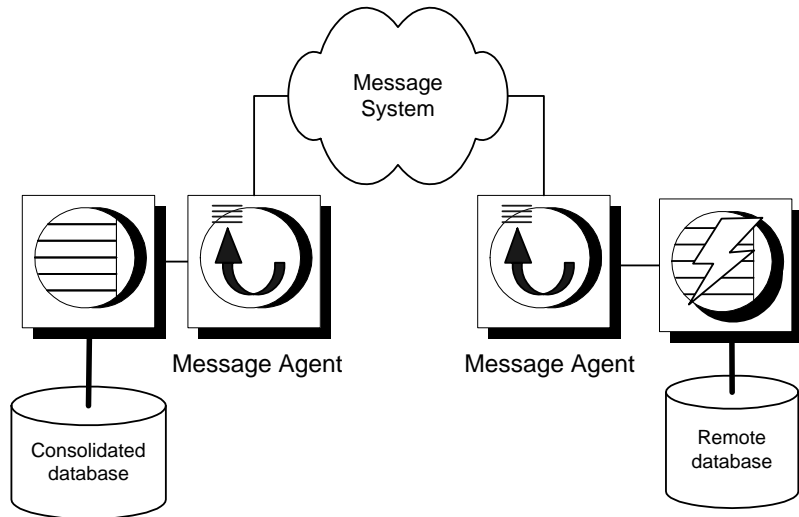
Client applications do not have to know if they are using a consolidated or remote database. From the client application perspective, there is no difference.

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

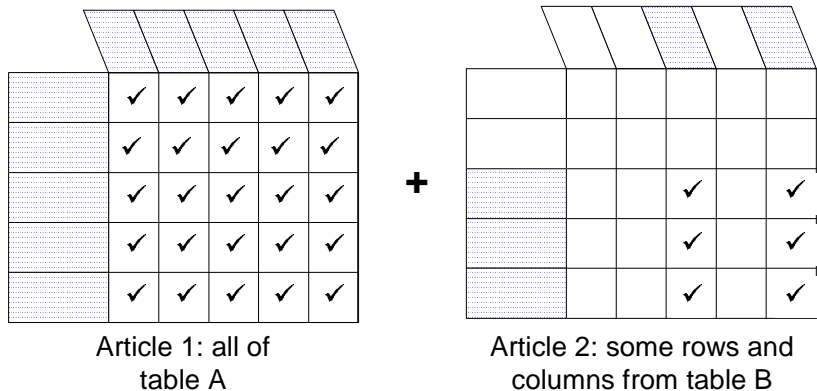
Data is organized into publications

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 synchronization definition



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

Messages are always sent both ways

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.

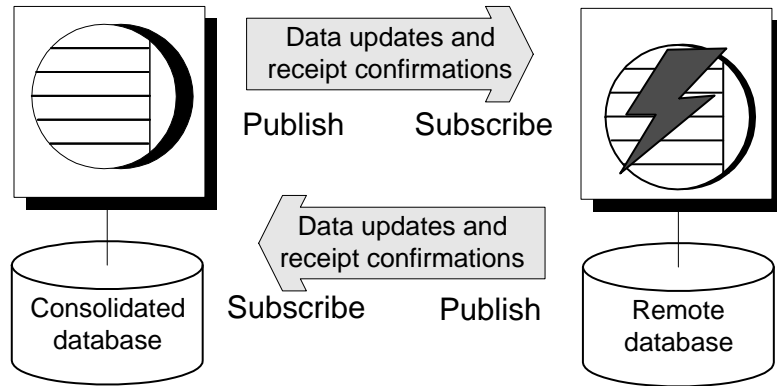
SQL Remote always involves messages being sent two ways. The consolidated database sends messages containing publication updates to remote databases, and remote databases also send messages to the consolidated database.

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

Both databases
subscribe

replication.

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.



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 as a command-line utility or, if you are using an Adaptive Server Anywhere consolidated database, from Sybase Central.

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-based 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 SQL Remote is provided on a number of operating systems and message links.

☞ For a list of supported environments, see [“Supported Platforms and Message Links” on page 443](#).

Some sample installations

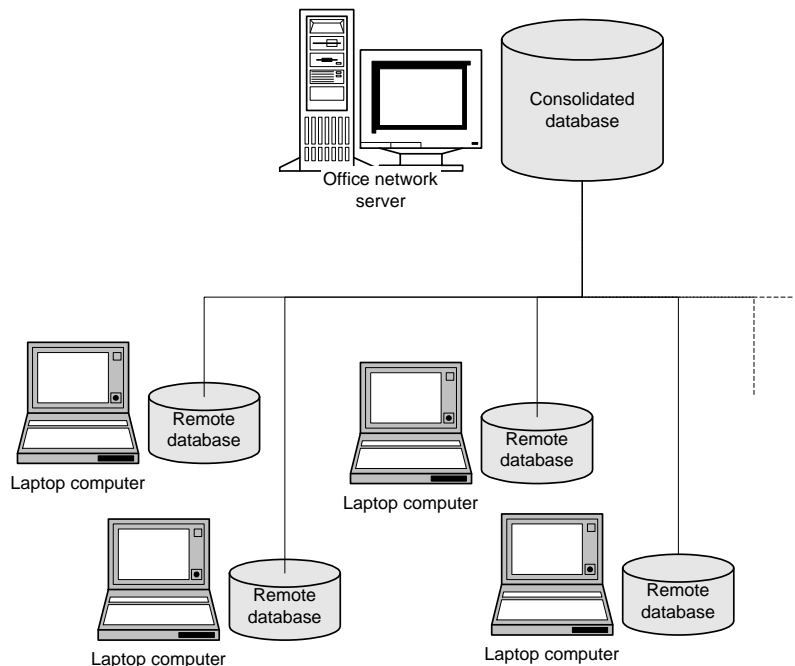
While SQL Remote can provide replication services in many different environments, its features are designed with the following characteristics in mind:

- ◆ 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.

At the laptop computers 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.

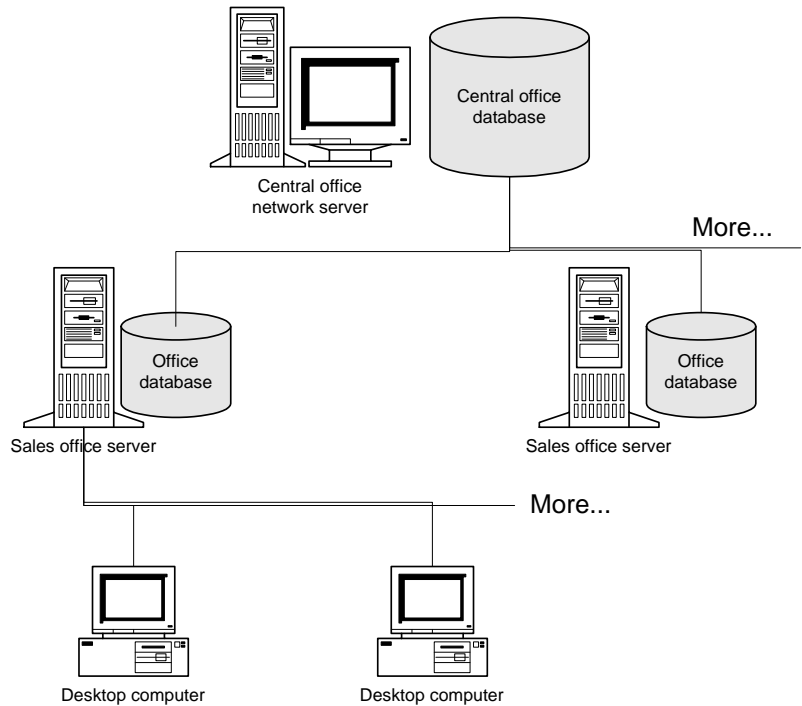
The new orders recorded by the sales representative are also automatically submitted to the office without any extra action on the part of the sales representative.

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

Topic:	page
Setup overview	20
Preparing your Adaptive Server Enterprise server	21
Upgrading SQL Remote for Adaptive Server Enterprise	25
Uninstalling SQL Remote	26

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.

Preparing 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 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*:

```
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:

```
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.

☞ The *login_id* must correspond to the name used by the Message Agent. For more information, see [“The Message Agent and replication security” on page 269](#).

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:

- ◆ **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 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 until they are no longer 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 265](#).

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*:

```
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.

☞ The *login_id* must correspond to the name used by the Message Agent. For more information, see [“The Message Agent and replication security” on page 269](#).

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

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.

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
```

CHAPTER 4

Tutorials for Adaptive Server Anywhere Users

About this chapter

This chapter guides you through setting up a simple replication system using Adaptive Server Anywhere.

Contents

Topic:	page
Introduction	28
Tutorial: Adaptive Server Anywhere replication using Sybase Central	32
Tutorial: Adaptive Server Anywhere replication using Interactive SQL and dbxtract	40
Start replicating data	47
A sample publication	51

Introduction

These tutorials describe how to set up a simple SQL Remote replication system using Adaptive Server Anywhere.

Goals

In the tutorials 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 tutorials take you through the following steps:

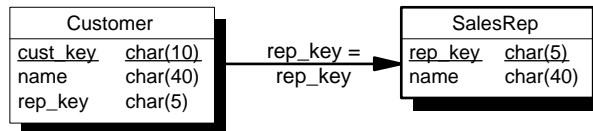
- ◆ Creating a consolidated database on your Adaptive Server Anywhere server.
- ◆ Creating a file-sharing replication system with a single Adaptive Server Anywhere remote database.
- ◆ Replicating data between the two databases.

The database

The tutorials use 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 tutorials 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 in to 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	Description
SalesRep	<p>One row for each sales representative that works for the company. The SalesRep table has the following columns:</p> <ul style="list-style-type: none"> ♦ rep_key An identifier for each sales representative. This is the primary key. ♦ name The name of each sales representative. <p>The SQL statement creating this table is as follows:</p> <pre>CREATE TABLE SalesRep (rep_key CHAR(12) NOT NULL, name CHAR(40) NOT NULL, PRIMARY KEY (rep_key))</pre>
Customer	<p>One row for each customer that does business with the company. The Customer table includes the following columns:</p> <ul style="list-style-type: none"> ♦ 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. <p>The SQL statement creating this table is as follows:</p> <pre>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))</pre>

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 tutorials describe 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 tutorial describes how to set up the installation using the Sybase Central management utility. The second tutorial 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 [“Tutorial: Adaptive Server Anywhere replication using Sybase Central”](#) on page 32.
- ◆ To work through the tutorial entering commands explicitly, go to [“Tutorial: Adaptive Server Anywhere replication using Interactive SQL and dbxtract”](#) on page 40.

Tutorial: Adaptive Server Anywhere replication 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 [“Tutorial: Adaptive Server Anywhere replication using Interactive SQL and dbxtract” on page 40](#). This tutorial 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. For an introduction to Sybase Central, see “Tutorial: Managing Databases with Sybase Central” [*Introducing SQL Anywhere Studio*, page 47].

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
```

3. Create the **HQ** database:

- ◆ Start Sybase Central.
- ◆ In the left pane, select the Adaptive Server Anywhere 9 plug-in.
- ◆ In the right pane, click the Utilities tab.
- ◆ Double-click Create Database in the right pane.
The Create Database wizard appears.
- ◆ Create a database with filename `c:\tutorial\HQ.db`.
Use the default settings for this database.

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, with a user ID of **DBA** and a password of **SQL**.
2. Select the Tables folder of the **HQ** database in the left pane.
3. From the File menu, choose New ► Table and create a table named **SalesRep** using the Table Creation wizard.
4. Add the following columns to the table (you can add a column by choosing File ► Add Column):

Key	Column	Data Type	Size/Prec
Primary key	Rep_key	char	5
	Name	char	40

You do not need to use the Column property sheet.

5. Save the table by choosing File ► Save Table or pressing Ctrl+S.
6. From the File menu, choose New ► Table and create a table named **Customer** with the following columns:

Key	Column	Data Type	Size/Prec
Primary key	Cust_key	char	10
	Name	char	40
	Rep_key	char	5

Again, you do not need to use the property sheets.

7. Save the table.
8. In the Tables folder in the left pane, select the Customer table, then click the Foreign Keys tab in the right pane.
9. From the File menu, choose New ► 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.
2. Open the SQL Remote Users folder for the **HQ** database.
3. In the right pane, click the Message Types tab.
4. In the right pane, right-click the FILE message type and choose Properties from the popup menu.
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 214](#).

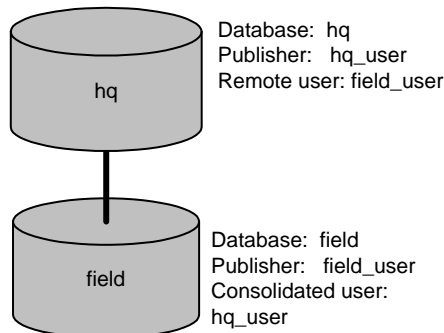
6. Click OK to save the message type.

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**.

Add a database
publisher user ID

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.

❖ **To create a new user as the publisher**

1. Select the Users & Groups folder.
2. From the File menu, choose New ► User.
The User Creation wizard appears.
3. Enter the name **hq_user**, with password **hq_pwd**, and click Finish.
4. Right-click the **hq_user** icon and choose Change to Publisher from the popup menu.

A database can have only one publisher. You can find out who the publisher is at any time by opening the Users & Groups 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 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.

❖ To add a remote user

1. Select the SQL Remote Users folder.
2. From the File menu, choose New ► SQL Remote User.
The Create a New Remote User wizard appears.
3. Create a remote user with user ID **field_user** with the following options:
 - ◆ Enter the password **field_pwd**.
 - ◆ Ensure that Remote DBA authority is selected, so that the user can run the Message Agent.
 - ◆ Select the message type **FILE**, and enter the address **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 214](#).
 - ◆ Ensure that the Send Then Close option is selected.
(In many production environments you would not choose Send Then Close, but it is convenient for this tutorial.)
4. When you have finished, 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. Select the Publications folder in the left pane.
2. From the File menu, choose New ► Publication.
The Publication Creation wizard appears.
3. Name the publication **SalesRepData**.
4. On the Tables tab, select **SalesRep** from the list of Available Tables. Click Add.
The table appears in the list of Selected Tables on the right.

-
5. Select **Customer** from the list of Available Tables. Click Add.
 6. On the SUBSCRIBE BY Restrictions tab, select the Customer table and enter the expression **rep_key**.
 7. Click Finish 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. Open the Publications folder and select the **SalesRepData** publication in the left pane.
2. Click the SQL Remote Subscriptions tab in the right pane.
3. From the File menu, choose New ► SQL Remote Subscription.

The SQL Remote Subscription Creation wizard appears.

4. Choose to subscribe **field_user**. Enter a Subscription value of **rep1** and click Finish.

The subscription value is an expression that matches the Subscribe By expression in the publication. In a later step, the **field_user** user ID is assigned a rep_key value of **rep1**.

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. Connect to the **HQ** database.
2. Right-click the database and choose Extract Database from the popup menu.
3. Choose to extract the HQ database with the following options:
 - ◆ Choose to extract at isolation level 3.
 - ◆ Choose to Start Subscriptions Automatically, for user **field_user**.
 - ◆ Leave the reload file location at its default setting and choose to extract both structure and data.
 - ◆ Leave the location to save the data at its default value.
 - ◆ Choose not to extract fully-qualified publication definitions.
 - ◆ Create the database as file `c:\tutorial\field.db`, and click Finish to 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.

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 47](#).

Tutorial: Adaptive Server Anywhere replication using Interactive SQL and dbxtract

The following sections are a tutorial describing how to set up a simple SQL Remote replication system for users 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 prompt:

```
dbinit hq.db  
dbinit field.db
```

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 prompt:


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

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

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

1. Connect to *hq.db* from Interactive SQL with a user ID of **DBA** and a password of **SQL**.
2. Execute the following CREATE TABLE statement to create the **SalesRep** table:

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

3. Execute the following CREATE TABLE statement to create the **Customer** table:

```
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 )
);
```

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

1. In Interactive SQL, 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 214](#).

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**.

GRANT PUBLISH to
identify outgoing
messages

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

1. 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
```

`GRANT REMOTE` for each database to which you send messages

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

1. Connect to the database using Interactive SQL, and execute 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 214.

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.

Set up a publication at the consolidated database

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

1. Connect to the database from Interactive SQL, and execute the following statement:

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

Set up a subscription

Each user ID that is to receive changes to the publication must have a subscription. The subscription can only be created for a user who has REMOTE permissions. The GRANT REMOTE statement contains the address to use when sending the messages.

❖ To create the subscription

1. Connect to the database from Interactive SQL, and execute the following statement:

```
CREATE SUBSCRIPTION  
TO SalesRepData ('repl')  
FOR field_user ;
```

The value **rep1** is the **rep_key** value we will give to the user **field_user** in the **SalesRep** table.

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 356](#).

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 "dbn=hq;uid=dba;pwd=sql" c:\tutorial field_user
```

The `-v` option 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 **dbn** database name parameter.

☞ For details of the *dbxtract* utility and its options, see [“The extraction utility” on page 303](#).

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 with a user ID of **DBA** and a password of **SQL**.
2. Run the *reload.sql* command file:

```
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 executing the following statements:

```
SELECT * FROM SalesRep ;
```

```
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 47](#) .

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 with a user ID of **DBA** and a password of **SQL**.
2. Insert two rows into the **SalesRep** table and commit the insertion by executing the following statement:

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

3. Insert two rows into the **Customer** table and commit the insertion by executing the following statement:

```
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 has been entered by executing the following statements:

```
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 "dbn=hq;uid=dba;pwd=sql"
```

This command line assumes that the **hq** database is currently running on the default server. If the database is not running, you must supply a **dbf** parameter with the database file name instead of the **dbn** parameter.

☞ For more information on *dbremote* options, see [“The Message Agent” on page 292](#).

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 "dbn=field;uid=dba;pwd=sql"
```

This command line assumes that the **field** database is currently running on the default server.

☞ For more information on *dbremote* options, see [“The Message Agent” on page 292](#).

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.

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 executing 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 **Customer** table by executing 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 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 by executing the following statement:

```
INSERT INTO Customer (cust_key, name, rep_key)
VALUES ('cust3', 'North Land Trading', 'repl')
```

3. Commit the insertion by executing the following statement::

```
COMMIT;
```

4. With the *field.db* database running, run the *dbremote* utility from a command line to send the message to the consolidated database.

```
dbremote -c "dbn=field;uid=dba;pwd=sql"
```

5. With the *hq.db* database running, run the *dbremote* utility from a command line to receive the message at the consolidated database:

```
dbremote -c "dbn=hq;uid=dba;pwd=sql"
```

6. Connect to the consolidated database. Display the **Customer** table by executing the following statement:

```
SELECT *  
FROM Customer
```

cust_key	name	rep_key
cust1	Ocean Sports	rep1
cust2	Sports Plus	rep2
cust3	North Land Trading	rep1

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 tutorials, in more detail.

❖ To add the publication to the sample database

1. Connect to the sample database from Interactive SQL.
2. In the SQL Statements pane, execute the following statement:

```
READ path\scripts\salespub.sql
```

where *path* is your SQL Anywhere 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.

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

Topic:	page
Introduction	54
Tutorial: Adaptive Server Enterprise replication	57
Start replicating data	66

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).

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:

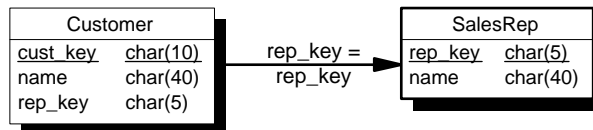
- ◆ Creating a consolidated database on your Adaptive Server Enterprise server.
- ◆ Creating a file-sharing replication system with a single Adaptive Server Anywhere remote database.
- ◆ Replicating data between the two databases.

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 in to 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	Description
SalesRep	<p>One row for each sales representative that works for the company. The SalesRep table has the following columns:</p> <ul style="list-style-type: none">♦ rep_key An identifier for each sales representative. This is the primary key.♦ name The name of each sales representative. <p>The SQL statement creating this table is as follows:</p> <pre>CREATE TABLE SalesRep (rep_key CHAR(12) NOT NULL, name CHAR(40) NOT NULL, PRIMARY KEY (rep_key))</pre>
Customer	<p>One row for each customer that does business with the company. The Customer table includes the following columns:</p> <ul style="list-style-type: none">♦ 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. <p>The SQL statement creating this table is as follows:</p> <pre>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))</pre>

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.

Tutorial: Adaptive Server Enterprise replication

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

This tutorial describes the stored procedures used to configure and manage SQL Remote. It also describes how to run the *ssxtract* 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 your password is **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 server 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.

Install SQL Remote

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

1. 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 214](#).

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.

❖ 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.

❖ 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
go
```

2. Add a user to the **hq** database:

```
exec sp_adduser field_user
go
```

3. Grant the user remote permissions. Execute the **sp_grant_remote** stored procedure, using **field_user** as the user name, **file** as the message type, and the appropriate directory as the address:

```
exec sp_grant_remote field_user, file, field
go
```

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.

☞ For information about setting the SQLRemote value, see [“Setting message type control parameters” on page 214](#).

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:

```
exec sp_add_article SalesRepData, SalesRep
go
```

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* 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 options have the following meaning.

- ◆ **-v** Verbose mode. For development work, this provides additional output.
- ◆ **-c** Connection string option. 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 options, see [“The extraction utility” on page 303](#).

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.

Creating an Adaptive
Server Anywhere
database

You can create an Adaptive Server Anywhere database using the *dbinit* 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*

1. Enter the following command from the *c:\tutorial\field* directory:

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

The *-b* option forces use of blank padding in string comparisons. The *-c* option enforces case sensitivity for string comparisons. The *-k* option 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* 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:

```
dbeng9 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:

```
dbeng9 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 66](#).

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 ('rep1', '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', 'rep1')
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

1. Enter the following statement (on a single line) at the command line to run the Message Agent against the consolidated database:

```
ssremote -c "eng=server-name;dbn=hq;uid=sa;pwd=sysadmin"
```

2. 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:

```
dbremote -c "eng=field;dbn=field;uid=DBA;pwd=SQL"
```

 For more information on *dbremote* options, see [“The Message Agent” on page 292](#).

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:

```
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:

```
SELECT * FROM Customer
```

You will see that the **Customer** table contains only one 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

```
INSERT INTO Customer (cust_key, name, rep_key)
VALUES ('cust3', 'North Land Trading', 'repl')
```

3. COMMIT the row.

```
COMMIT;
```

4. With the *field.db* database running, run *dbremote* to send the message to the consolidated database.

```
dbremote -c "eng=field;dbn=field;uid=DBA;pwd=SQL"
```

5. Run *ssremote* to receive the message at the consolidated database:

```
ssremote -c "eng=server-name;dbn=hq;uid=sa;pwd=sysadmin"
```

6. Connect to the consolidated database and display the **Customer** table.
This now has three rows:

```
SELECT *  
FROM Customer
```

cust_key	name	rep_key
cust1	Ocean Sports	rep1
cust2	Sports Plus	rep2
cust3	North Land Trading	rep1

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.

PART II

REPLICATION DESIGN FOR SQL REMOTE

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 141 and “SQL Remote Design for Adaptive Server Anywhere” on page 91.

Contents

Topic:	page
Design overview	74
How statements are replicated	78
How data types are replicated	83
Who gets what?	86
Replication errors and conflicts	88

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.

Design at the consolidated database

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

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

Ensuring compatible databases

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

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

☞ For a full description of how to create Enterprise-compatible Adaptive Server Anywhere databases, see “Creating a Transact-SQL-compatible database” [*ASA SQL User’s Guide*, page 449]. This section provides a brief description only.

❖ To create an Enterprise-compatible Adaptive Server Anywhere database (Sybase Central)

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

❖ To create an Enterprise-compatible Adaptive Server Anywhere database (Command line)

1. **Ensure trailing blanks are ignored** You can do this using the `dbinit -b` option.
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` option.
3. **Remove historical system views** You can do this with the `dbinit -k` option.

4. **Make the database case sensitive** You can do this with the `dbinit -c` option.

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

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

Using compatible sort orders and character sets

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

Character sets in
Adaptive Server
Anywhere installations

For an Adaptive Server Anywhere installation, the character set and collation used by the consolidated database must be the same as the remote databases. For information about supported character sets, see “International Languages and Character Sets” [ASA Database Administration Guide, page 285].

Character sets in
Adaptive Server
Enterprise installations

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

Character sets in mixed
installations

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

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

Adaptive Server Anywhere colla- tion name	Open Client / Open Server name	Open Client / Open Server case- sensitive sort order	Open Client / Open Server case- insensitive sort order
default	cp850	dictionary_cp850	nocase_cp850
437LATIN1	cp437	dictionary_cp437	nocase_cp437
437ESP	cp437	espdict_cp437	espnocs_cp437
437SVE	cp437	bin_cp437	bin_cp437
819CYR	iso_1	bin_iso_1	bin_iso_1
819DAN	iso_1	bin_iso_1	bin_iso_1

Adaptive Server Anywhere colla- tion name	Open Client / Open Server name	Open Client / Open Server case- sensitive sort order	Open Client / Open Server case- insensitive sort order
819ELL	iso_1	bin_iso_1	bin_iso_1
819ESP	iso_1	espdict_iso_1	espnocs_iso_1
819ISL	iso_1	bin_iso_1	bin_iso_1
819LATIN1	iso_1	dictionary_iso_1	nocase_iso_1
819LATIN2	iso_1	bin_iso_1	bin_iso_1
819NOR	iso_1	bin_iso_1	bin_iso_1
819RUS	iso_1	bin_iso_1	bin_iso_1
819SVE	iso_1	bin_iso_1	bin_iso_1
819TRK	iso_1	bin_iso_1	bin_iso_1
850CYR	cp850	bin_cp850	bin_cp850
850DAN	cp850	scandict_cp850	scannocp_- cp850
850ELL	cp850	bin_cp850	bin_cp850
850ESP	cp850	espdict_cp850	espnocs_cp850
850ISL	cp850	scandict_cp850	scannocp_- cp850
850LATIN1	cp850	dictionary_cp850	nocase_cp850
850LATIN2	cp850	bin_cp850	bin_cp850
850NOR	cp850	scandict_cp850	scannocp_- cp850
850RUS	cp850	bin_cp850	bin_cp850
850SVE	cp850	scandict_cp850	scannocp_- cp850
850TRK	cp850	bin_cp850	bin_cp850
852LATIN2	cp852	bin_cp852	bin_cp852
852CYR	cp852	bin_cp852	bin_cp852

Adaptive Server Anywhere colla- tion name	Open Client / Open Server name	Open Client / Open Server case- sensitive sort order	Open Client / Open Server case- insensitive sort order
855CYR	cp855	cyrdict_cp855	cynocs_cp855
857TRK	cp857	bin_cp857	bin_cp857
860LATIN1	cp860	bin_cp860	bin_cp860
866RUS	cp866	rusdict_cp866	rusnocs_cp866
869ELL	cp869	bin_cp869	bin_cp869
932JPN	sjis	bin_sjis	bin_sjis
EUC_JAPAN	eucjis	bin_eucjis	bin_eucjis
EUC_CHINA	eucgb	bin_eucgb	bin_eucgb
EUC_TAIWAN	eucb5	bin_big5	bin_big5
EUC_KOREA	eucksc	bin_eucksc	bin_eucksc
UTF8	utf8	bin_utf8	bin_utf8

How statements are replicated

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

Replication of inserts and deletes

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

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

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

Replication of updates

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

UPDATE statements
replicated as INSERTS
or DELETES

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:

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

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

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

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

UPDATE conflict
detection

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

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

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

In this case, the UPDATE does not update anything.

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

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

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

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

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

Replication of procedures

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

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

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

Replication of triggers

Trigger replication 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

From Adaptive Server Enterprise, trigger actions are replicated. You must ensure that triggers are not fired in the remote Adaptive Server Anywhere databases. If the trigger were fired, its actions would be executed twice.

The Adaptive Server Anywhere FIRE_TRIGGERS database option prevents triggers from being fired. If you set this option for the user ID used by the Message Agent, be careful to not use this user ID for other purposes.

An alternative approach to preventing trigger execution, available only for Adaptive Server Anywhere, is to use the following condition around the body of your triggers:

```
IF CURRENT_REMOTE_USER IS NULL
```

This makes execution conditional on whether the current user is 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 an option that causes it to replicate all trigger actions when sending messages. This is the *dbremote -t* option.

If you use this option, 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 260](#).

How data types are replicated

Long binary or character data, and datetime data, need special consideration.

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

Blobs can be replicated to and from Adaptive Server Enterprise as long as they fit into the Message Agent memory.

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

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 200 kb in length, this causes 4 Mb of messages to be sent.

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 (200 kb of messages).

Controlling replication of blobs

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.

Replication of dates and times

When date or time columns are replicated, the Message Agent uses the setting of the SR_Date_Format, SR_Time_Format, and SR_Timestamp_Format database options to format the date.

For example, the following option setting instructs the Message Agent to send a date of May 2, 1987 as 1987-05-02.

```
SET OPTION SR_Date_Format = 'yyyy-mm-dd'
```

☞ For more information, see [“SQL Remote options” on page 313](#).

☞ The following points may be useful when replicating dates and times:

- ◆ The time, date, and timestamp formats must be consistent throughout the installation.
- ◆ If the consolidated database is an Adaptive Server Anywhere database, ensure that the order of year, month, and day used for the date and timestamp formats matches the setting of the DATE_ORDER database option.

You can change the DATE_ORDER option for the duration of each connection.

- ◆ If the consolidated database is an Adaptive Server Enterprise database, ensure that the order of year, month, and day in the SQL Remote settings is consistent with the dateformat setting in the Adaptive Server Enterprise database.

❖ **To find the dateformat settings on an Adaptive Server Enterprise database**

1. Login to the Adaptive Server Enterprise database from *isql* using the login ID used by *ssremote*. In this example, we use **ssr** for this login ID.
2. Issue the following command:

```
select *  
from master..syslogins  
where name = 'ssr'  
go
```

Adaptive Server Enterprise returns the default language for the *ssr* user.

3. If *ssr* uses the default language (*us_english*) then the default dateformat is *YMD*. If the language is different from the default, enter the following command:

```
sp_helplanguage language-name
```

where *language-name* is the language in use by the *ssr* user. The information displayed includes the default date format for the language.

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 78](#). For details on subscriptions, see the following two chapters.

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

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

Adaptive Server
Anywhere actions

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

Not the subscriber list

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

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

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

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

Adaptive Server
Enterprise actions

In a SQL Remote for Adaptive Server Enterprise publication, the subscription expression must be a column. The subscription column

contains either a single value or a comma-separated list of values.

Not the subscriber list

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

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

Message Agent actions

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

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

Replication errors and conflicts

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

Delivery errors not discussed here

This section does not discuss issues related to message delivery failures. For information on delivery errors and how they are handled, see [“The message tracking system” on page 237](#)

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

1. 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

1. Issue the following statement:

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

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

Replication error
procedure requirements

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

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 141.

Contents

Topic:	page
Design overview	92
Publishing data	93
Publication design for Adaptive Server Anywhere	102
Partitioning tables that do not contain the subscription expression	105
Sharing rows among several subscriptions	112
Managing conflicts	120
Ensuring unique primary keys	129
Creating subscriptions	139

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.

Publishing data

This section describes how to create simple publications consisting of whole tables, or of column-wise subsets of tables; these tables are also called articles. You can perform these tasks using Sybase Central or with the `CREATE PUBLICATION` statement in Interactive SQL.

All publications in Sybase Central appear in the Publications folder. Any articles you create for a publication appear on the Articles tab in the right pane when a publication is selected.

Each publication can contain one or more entire tables, but partial tables are also permitted. A table can be subdivided by columns, rows, or both.

Publishing whole tables

The simplest publication you can make consists of a single article, which consists of all rows and columns of one or more tables. These tables must already exist.

❖ To publish one or more entire tables (Sybase Central)

1. Connect to the database as a user with DBA authority.
2. In the left pane, select the Publications folder.
3. From the File menu, choose New ► Publication.
The Publication Creation wizard appears.
4. Type a name for the publication. Click Next.
5. On the Tables tab, select a table from the list of Available tables.
Click Add. The table appears in the list of Selected Tables on the right.
6. Optionally, you may add additional tables. The order of the tables is not important.
7. Click Finish.

❖ To publish one or more entire tables (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a CREATE PUBLICATION statement that specifies the name of the new publication and the table you want to publish.

Example

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

```
CREATE PUBLICATION pub_customer (  
    TABLE customer  
)
```

- ◆ 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  
)
```

☞ For more information, see the “CREATE PUBLICATION statement” [ASA SQL Reference, page 334].

Publishing only some columns in a table

You can create a publication that contains all the rows, but only some of the columns, of a table from Sybase Central or by listing the columns in the CREATE PUBLICATION statement.

❖ To publish only some columns in a table (Sybase Central)

1. Connect to the database as a user with DBA authority.
2. In the left pane, select the Publications folder.
3. From the File menu, choose New ► Publication.
The Publication Creation wizard appears.
4. Type a name for the new publication. Click Next.
5. On the Tables tab, select a table from the list of Available tables. Click Add. The table is added to the list of Selected Tables on the right.

6. On the Columns tab, double-click the table's icon to expand the list of Available Columns. Select each column you want to publish and click Add. The selected columns appear on the right in the Selected Columns list.
7. Click Finish.

❖ **To publish only some columns in a table (SQL)**

1. Connect to the database as a user with DBA authority.
 2. Execute a CREATE PUBLICATION statement that specifies the publication name and the table name. List the published columns in parenthesis following the table 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,  
        company_name,  
        city )  
    )
```

☞ For more information, see the “CREATE PUBLICATION statement” [ASA SQL Reference, page 334].

Publishing only some rows in a table

You can create a publication that contains all the columns, but only some of the rows, of a table from Sybase Central. In either case, you do so by writing a search condition that matches only the rows you want to publish.

Sybase Central and the SQL language provide two ways of publishing only some of the rows in a table; however, only one way is compatible with MobiLink.

- ◆ **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.

You can combine a WHERE clause and a subscription expression in an article. You can specify them in Sybase Central or in a CREATE PUBLICATION statement.

Use the 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.

Use the WHERE clause to exclude the same set of rows from all subscriptions to a publication.

Publishing only some rows using a WHERE clause

You can specify a WHERE clause to include in the publication only the rows that satisfy the WHERE conditions.

❖ To create a publication using a WHERE clause (Sybase Central)

1. Connect to the database as a user with DBA authority.
2. In the left pane, select the Publications folder.
3. From the File menu, choose New ► Publication.
The Publication Creation wizard opens.
4. Type a name for the new publication. Click Next.
5. On the Tables tab, select a table from the list of Available tables. Click Add. The table is added to the list of Selected Tables on the right.
6. On the WHERE Clauses tab, select the table then type the search condition in the lower box.
7. Click Finish.

❖ To create a publication using a WHERE clause (SQL)

1. Connect to the database as a user with DBA authority.
 2. Execute a CREATE PUBLICATION statement that includes the rows you wish to include in the publication and a WHERE condition.
- ◆ 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'  
)
```

Examples

In this case, the status column is not published. All unpublished rows must have a default value. Otherwise, an error occurs when rows are downloaded for insert from the consolidated database.

- ◆ 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
)
```

☞ For more information, see the “CREATE PUBLICATION statement” [ASA SQL Reference, page 334].

SUBSCRIBE BY

The create publication statement also allows a SUBSCRIBE BY clause. This clause can also be used to selectively publish rows in SQL Remote. However, it is ignored during MobiLink synchronization.

Publishing only some rows using a subscription expression

You can specify a subscription expression to include a different set of rows in different subscriptions to publications containing the article.

For example, 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.

```
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 (Sybase Central)**

1. Connect to the database as a user with DBA authority.
2. In the left pane, select the Publications folder.
3. From the File menu, choose New ► Publication.
The Publication Creation wizard appears.
4. Type a name for the publication and click Next.
5. On the Tables tab, configure the desired values for that table.
6. On the SUBSCRIBE BY Restrictions tab, use the controls to create the subscription expression.
7. Follow the remaining instructions in the wizard.

❖ **To create an article using a subscription expression (SQL)**

1. Connect to the database as a user with DBA authority.
 2. Execute a CREATE PUBLICATION statement that includes the expression you wish to use as a match in the subscription expression.
- ◆ 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:

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

- ◆ 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).

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

Examples

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

☞ See also

- ◆ “CREATE PUBLICATION statement” [ASA SQL Reference, page 334]
- ◆ [“Partitioning tables that do not contain the subscription expression” on page 105](#)
- ◆ [“Creating subscriptions” on page 139](#)
- ◆ [“Publishing only some rows using a WHERE clause” on page 96](#)
- ◆ [“Altering existing publications” on page 99](#)

Altering existing publications

After you have created a publication, you can alter it 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.

You can perform these tasks using Sybase Central or with the ALTER PUBLICATION statement in Interactive SQL.

❖ To modify the properties of existing publications or articles (Sybase Central)

1. Connect to the database as a user who owns the publication or as a user with DBA authority.
2. Right-click the publication or article and choose Properties from the popup menu.
3. Configure the desired properties.

❖ To add articles (Sybase Central)

1. Connect to the database as a user who owns the publication or as a user with DBA authority.
2. In the left pane, open the Publications folder.
3. Select the publication you want to add an article to.
4. From the File menu, choose New ► Article.
The Article Creation wizard appears.
5. In the Article Creation wizard, do the following:

- ◆ Choose a table and click Next.
- ◆ Choose the columns for the article. Click Next.
- ◆ Enter a WHERE clause (if desired). Click Next.
- ◆ Create a SUBSCRIBE BY restriction (if desired).

6. Click Finish to create the article.

❖ To remove articles (Sybase Central)

1. Connect to the database as a user who owns the publication or as a user with DBA authority.
2. Open the Publications folder.
3. Select the publication you want to remove an article from.
4. Right-click the article you want to delete and choose Delete from the popup menu.

❖ To modify an existing publication (SQL)

1. Connect to the database as a user who owns the publication or as a user with DBA authority.
2. Connect to a database with DBA authority.
3. Execute an ALTER PUBLICATION statement.

Example

- ◆ The following statement adds the customer table to the pub_contact publication.

```
ALTER PUBLICATION pub_contact (
    ADD TABLE customer
)
```

☞ See also

- ◆ “ALTER PUBLICATION statement” [ASA SQL Reference, page 238]
- ◆ “Publishing only some rows using a WHERE clause” on page 96
- ◆ “Publishing only some rows using a subscription expression” on page 97

Dropping publications

You can drop a publication using either Sybase Central or the DROP PUBLICATION statement. If you drop a publication, all subscriptions to that publication are automatically deleted as well.

You must have DBA authority to drop a publication.

❖ **To delete a publication (Sybase Central)**

1. Connect to the database as a user with DBA authority.
2. Open the Publications folder.
3. Right-click the desired publication and choose Delete from the popup menu.

❖ **To delete a publication (SQL)**

1. Connect to the database as a user with DBA authority.
2. Execute a DROP PUBLICATION statement.

Example

The following statement drops the publication named pub_orders.

```
DROP PUBLICATION pub_orders
```

☞ See also the “DROP PUBLICATION statement” [ASA SQL Reference, page 413].

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 a WHERE clause to select a set of rows to be replicated.
- ◆ DBA authority is required to create and drop publications.
- ◆ Publications can be altered only by the DBA or the publication’s owner.
- ◆ Altering publications in a running SQL Remote setup is likely to cause replication errors and can 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 80](#).

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 147](#).

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

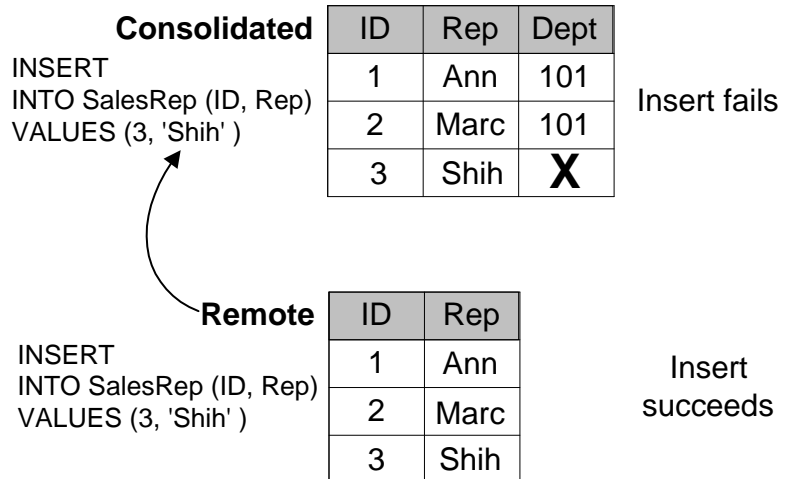
All columns in the primary key must be included in the article.

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.



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 checklist 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

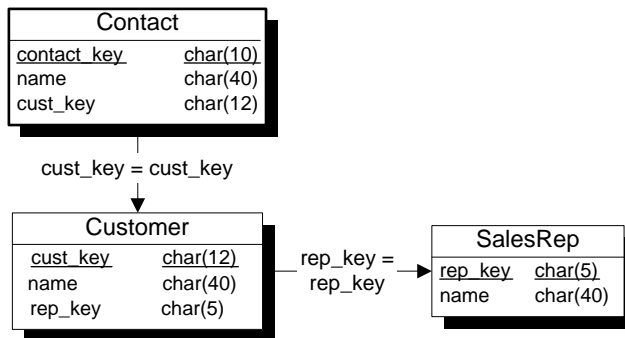
In many cases, the rows of a table need to be partitioned even when the subscription expression does not exist in the table.

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.



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:

Table	Description
SalesRep	<p>All sales representatives that work for the company. The SalesRep table has the following columns:</p> <ul style="list-style-type: none"> ♦ rep_key An identifier for each sales representative. This is the primary key. ♦ name The name of each sales representative. <p>The SQL statement creating this table is as follows:</p> <pre> CREATE TABLE SalesRep (Rep_key CHAR(12) NOT NULL, Name CHAR(40) NOT NULL, PRIMARY KEY (rep_key)) </pre>

Table	Description
Customer	<p>All customers that do business with the company. The Customer table includes the following columns:</p> <ul style="list-style-type: none"> ♦ 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. <p>The SQL statement creating this table is as follows:</p> <pre>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))</pre>
Contact	<p>All individual contacts that do business with the company. Each contact belongs to a single customer. The Contact table includes the following columns:</p> <ul style="list-style-type: none"> ♦ 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. <p>The SQL statement creating this table is:</p> <pre>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))</pre>

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:

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

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 Contact.cust_key = Customer.cust_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 149](#).

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 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 86](#).

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:
- ◆ 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 be 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 112](#), 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

rep_key	Name
rep1	Ann
rep2	Marc

- Customer table

cust_key	name	rep_key
cust1	Sybase	rep1
cust2	ASA	rep2

- Contact table

contact_key	name	cust_key
contact1	David	cust1
contact2	Stefanie	cust2

- ◆ Now apply the following territory realignment Update statement

```
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.

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

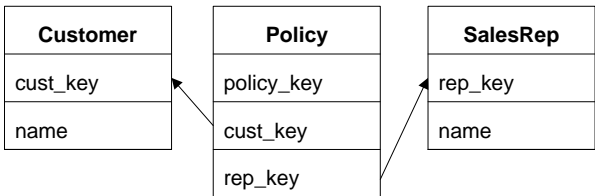
There are cases where a row may need to be included in several subscriptions. For example, we may have a many-to-many relationship. In this section, we use a case study to illustrate how to handle this situation.

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.



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	Description
SalesRep	<p>All sales representatives that work for the company. The SalesRep table has the following columns:</p> <ul style="list-style-type: none">♦ rep_key An identifier for each sales representative. This is the primary key.♦ name The name of each sales representative. <p>The SQL statement creating this table is as follows:</p> <pre>CREATE TABLE SalesRep (Rep_key CHAR(12) NOT NULL, Name CHAR(40) NOT NULL, PRIMARY KEY (rep_key));</pre>

Table	Description
Customer	<p>All customers that do business with the company. The Customer table includes the following columns:</p> <ul style="list-style-type: none"> ♦ cust_key A primary key column containing an identifier for each customer ♦ name A column containing the name of each customer <p>The SQL statement creating this table is as follows:</p> <pre>CREATE TABLE Customer (Cust_key CHAR(12) NOT NULL, Name CHAR(40) NOT NULL, PRIMARY KEY (cust_key));</pre>
Policy	<p>A three-column table that maintains the many-to-many relationship between customers and sales representatives. The Policy table has the following columns:</p> <ul style="list-style-type: none"> ♦ policy_key A primary key column containing an identifier for the sales relationship. ♦ cust_key A column containing an identifier for the customer representative in a sales relationship. ♦ rep_key A column containing an identifier for the sales representative in a sales relationship. <p>The SQL statement creating this table is as follows.</p> <pre>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));</pre>

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.

New problems

- ◆ Those rows from the **Customer** table listing customers that deal with the sales rep subscribed to the data.

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.

Put another way, the rows of the **Contact** table in “[Partitioning tables that do not contain the subscription expression](#)” on page 105 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:

```
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 107](#).

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.

```
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:

```
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 Policy_key <> OldRow.Policy_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

Multiple-valued
subqueries

that this subquery can be multi-valued.

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

```
...
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:

```
...
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

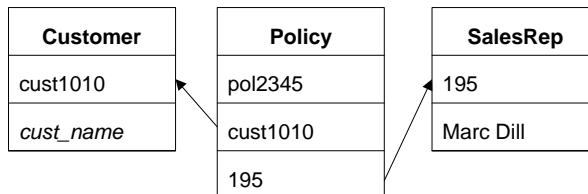
When the `Subscribe_by_remote` option is ON, operations from remote databases on rows with a subscribe by value of NULL or an empty string will assume the remote user is subscribed to the row. By default, the `Subscribe_by_remote` option is set to ON. In most cases, this setting is the desired setting.

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:

```
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.

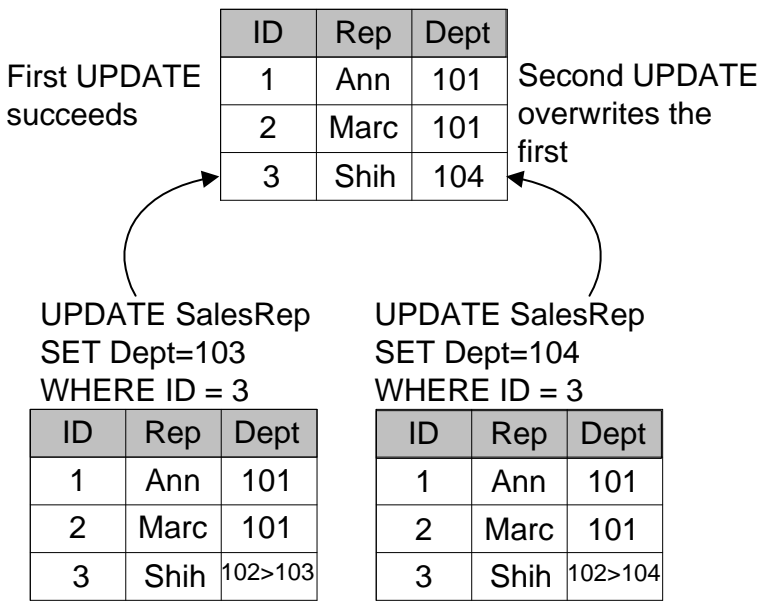
If Subscribe_by_remote is set to OFF, you must ensure that the Policy row is inserted before the Customer row, with the referential integrity violation avoided by postponing checking to the end of the transaction.

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.

☞ 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 166](#).

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:

```

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:

```

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.

You can set the VERIFY_ALL_COLUMNS option either for the PUBLIC

Using the CURRENT
REMOTE USER special
constant

group or just for the user contained in the Message Agent connection string.

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.

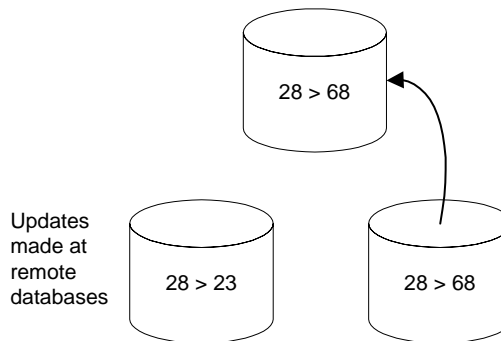
```
CREATE TRIGGER contact_date RESOLVE UPDATE
ON contact
REFERENCING OLD AS old_name
NEW AS new_name
FOR EACH ROW
BEGIN
    IF new_name.contact_date <
        old_name.contact_date THEN
        SET new_name.contact_date
            = old_name.contact_date
    END IF
END
```

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

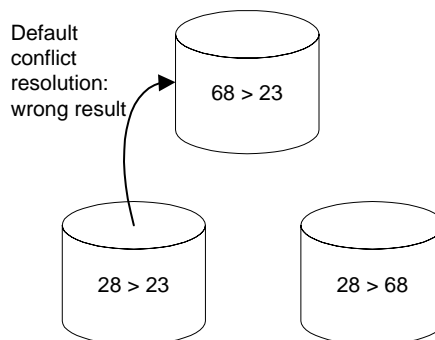
Consider a warehouse system for a manufacturer of sporting goods. There is a table of product information, with a **quantity** column holding the number of each product left in stock. An update to this column will typically deplete the quantity in stock or, if a new shipment is brought in, add to it.

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.



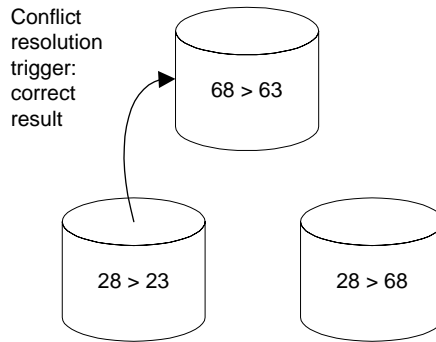
The warehouse entry gets added to the database: the **quantity** column now shows there are 68 small tank-top tee shirts in stock. When the update from the sales representative arrives, it causes a conflict—Adaptive Server Anywhere detects that the update is from 28 to 23, but that the current value of the column is 68.

By default, the most recent UPDATE succeeds, and the inventory level is set to the incorrect value of 23.



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,

```
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.

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.

Unreplicated referenced
table errors

The **sales** publication described in [“Publishing whole tables” on page 93](#) includes the **sales_order** table:

```
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 124](#). The actions of RESOLVE UPDATE triggers are replicated to remote databases, including the database that caused the conflict.

Ensuring unique primary keys

Primary key values must be unique. When all users are connected to the same database, there is no problem keeping unique values. If a user tries to re-use a value, the INSERT statement fails.

The situation is different in a replication system because users are connected to many databases. A potential problem arises when two users, connected to different databases, insert a row using the same primary key value. Each of their statements succeeds because the value is unique in each database.

However, problems arise in a replication system when two users, connected to separate databases, INSERT a row using the same primary key value. The second INSERT to reach a given database in the replication system fails. 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 duplication 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 two general, economical, and reliable methods.

1. Using the default global autoincrement feature of Adaptive Server Anywhere.
2. Using the primary key pools to maintain a list of unused, unique primary key values at each site.

You can use these techniques either separately or together to avoid duplicate values.

Using global autoincrement default column values

In Adaptive Server Anywhere, you can set the default column value to be GLOBAL AUTOINCREMENT. You can use this default for any column in which you want to maintain unique values, but it is particularly useful for primary keys. This feature is intended to simplify the task of generating unique values in setups where data is being replicated among multiple databases, typically by MobiLink synchronization.

When you specify default global autoincrement, the domain of values for that column is partitioned. Each partition contains the same number of values. For example, if you set the partition size for an integer column in a database to 1000, one partition extends from 1001 to 2000, the next from 2001 to 3000, and so on.

You assign each copy of the database a unique global database identification number. Adaptive Server Anywhere supplies default values in a database only from the partition uniquely identified by that database's number. For example, if you assigned the database in the above example the identity number 10, the default values in that database would be chosen in the range 10001–11000. Another copy of the database, assigned the identification number 11, would supply default value for the same column in the range 11001–12000.

Declaring default global autoincrement

You can set default values in your database by selecting the column properties in Sybase Central, or by including the `DEFAULT GLOBAL AUTOINCREMENT` phrase in a `TABLE` or `ALTER TABLE` statement.

Optionally, the partition size can be specified in parentheses immediately following the `AUTOINCREMENT` keyword. The partition size may be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, be exhausted.

For columns of type `INT` or `UNSIGNED INT`, the default partition size is $2^{16} = 65536$; for columns of other types the default partition size is $2^{32} = 4294967296$. Since these defaults may be inappropriate, especially if our column is not of type `INT` or `BIGINT`, it is best to specify the partition size explicitly.

For example, the following statement creates a simple table with two columns: an integer that holds a customer identification number and a character string that holds the customer's name.

```
CREATE TABLE customer (  
    id    INT          DEFAULT GLOBAL AUTOINCREMENT (5000),  
    name  VARCHAR(128) NOT NULL,  
    PRIMARY KEY (id)  
)
```

In the above example, the chosen partition size is 5000.

☞ For more information on `GLOBAL AUTOINCREMENT`, see “`CREATE TABLE` statement” [*ASA SQL Reference*, page 361].

Setting the Global_database_id value

When deploying an application, you must assign a different identification number to each database. You can accomplish the task of creating and distributing the identification numbers by a variety of means. One method is to place the values in a table and download the correct row to each database based on some other unique property, such as user name.

❖ **To set the global database identification number**

1. You set the identification number of a database by setting the value of the public option **Global_database_id**. The identification number must be a non-negative integer.

For example, the following statement sets the database identification number to 20.

```
SET OPTION PUBLIC.Global_database_id = 20
```

If the partition size for a particular column is 5000, default values for this database are selected from the range 100001–105000.

Setting unique database identification numbers when extracting databases

If you use the extraction utility to create your remote databases, you can write a stored procedure to automate the task. If you create a stored procedure named `sp_hook_dbxtract_begin`, it is called automatically by the extraction utility. Before the procedure is called, the extraction utility creates a temporary table named `#hook_dict`, with the following contents:

name	value
extracted_db_global_id	user ID being extracted

If you write your `sp_hook_dbxtract_begin` procedure to modify the value column of the row, that value is used as the `GLOBAL_DATABASE_ID` option of the extracted database, and marks the beginning of the range of primary key values for `GLOBAL DEFAULT AUTOINCREMENT` values.

Example

Consider extracting a database for remote user **user2** with a **user_id** of 101. If you do not define an `sp_hook_dbxtract_begin` procedure, the extracted database will have **Global_database_id** set to **101**.

If you define a `sp_hook_dbxtract_begin` procedure, but it does not modify any rows in the `#hook_dict` then the option will still be set to **101**.

If you set up the database as follows:

```

set option "PUBLIC"."Global_database_id" = '1';
create table extract_id ( next_id integer not null ) ;
insert into extract_id values( 1 );
create procedure sp_hook_dbxtract_begin
as
    declare @next_id  integer
    update extract_id set next_id = next_id + 1000
    select @next_id = (next_id )
    from extract_id
    commit
    update #hook_dict
    set value = @next_id
    where name = 'extracted_db_global_id'

```

Then each extracted or re-extracted database will get a different **Global_database_id**. The first starts at 1001, the next at 2001, and so on.

To assist in debugging procedure hooks, *dbxtract* outputs the following when it is set to operate in verbose mode:

- ◆ the procedure hooks found
- ◆ the contents of #hook_dict before the procedure hook is called
- ◆ the contents of #hook_dict after the procedure hook is called.

How default values are chosen

The public option **Global_database_id** in each database must be set to a unique, non-negative integer. The range of default values for a particular database is $pn + 1$ to $p(n + 1)$, where p is the partition size and n is the value of the public option **Global_database_id**. For example, if the partition size is 1000 and **Global_database_id** is set to 3, then the range is from 3001 to 4000.

If **Global_database_id** is set to a non-negative integer, Adaptive Server Anywhere chooses default values by applying the following rules:

- ◆ If the column contains no values in the current partition, the first default value is $pn + 1$.
- ◆ If the column contains values in the current partition, but all are less than $p(n + 1)$, the next default value will be one greater than the previous maximum value in this range.
- ◆ Default column values are not affected by values in the column outside of the current partition; that is, by numbers less than $pn + 1$ or greater than $p(n + 1)$. Such values may be present if they have been replicated from another database via MobiLink synchronization.

If the public option **Global_database_id** is set to the default value of 2147483647, a null value is inserted into the column. Should null values not be permitted, the attempt to insert the row causes an error. This situation arises, for example, if the column is contained in the table's primary key.

Because the public option **Global_database_id** cannot be set to negative values, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

Null default values are also generated when the supply of values within the partition has been exhausted. In this case, a new value of **Global_database_id** should be assigned to the database to allow default values to be chosen from another partition. Attempting to insert the null value causes an error if the column does not permit nulls. To detect that the supply of unused values is low and handle this condition, create an event of type **GlobalAutoincrement**.

Should the values in a particular partition become exhausted, you can assign a new database id to that database. You can assign new database id numbers in any convenient manner. However, one possible technique is to maintain a pool of unused database id values. This pool is maintained in the same manner as a pool of primary keys.

You can set an event handler to automatically notify the database administrator (or carry out some other action) when the partition is nearly exhausted. For more information, see “Defining trigger conditions for events” [ASA Database Administration Guide, page 273].

☞ For more information, see “GLOBAL_DATABASE_ID option [database]” [ASA Database Administration Guide, page 595].

☞ For further information on pools, see [“Using primary key pools” on page 133](#).

Using primary key pools

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 table

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 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:

```
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 (SQL)

1. Create a publication for the primary key pool data.

```
CREATE PUBLICATION KeyPoolData (
    TABLE KeyPool SUBSCRIBE BY location
);
```

2. Create subscriptions for each remote database to the KeyPoolData publication.

```
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.

☞ See also

- ◆ “CREATE PUBLICATION statement” [ASA SQL Reference, page 334]
- ◆ “CREATE SUBSCRIPTION statement” on page 356

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:

```

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;

```

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.
        // 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.

```

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.

Working with
subscriptions in Sybase
Central

❖ To create and manage subscriptions in Sybase Central

1. In the left pane, open the Publications folder.
2. Select the desired publication. You can perform the following tasks in Sybase Central:
3. In the right pane, click the SQL Remote Subscriptions tab. You can configure the appropriate settings as follows:
 - ◆ To subscribe a remote user to the publication, from the File menu choose New ► SQL Remote Subscription and follow the instructions in the SQL Remote Subscription Creation wizard.
 - ◆ To unsubscribe a remote user, right-click the user in the Subscribers list and choose Delete from the popup menu.
 - ◆ To manually start, stop, or synchronize subscriptions, select the user in the Subscribers list and choose Properties from the popup menu. Click the Advanced tab. On this tab, click Start Now to start subscriptions, Stop Now to stop subscriptions, or Synchronize Now to synchronize subscriptions. The subscriptions are affected as soon as you click the button. Subsequently clicking Cancel on the property sheet does *not* cancel your start/stop/synchronize action.

Subscriptions with no
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:

```
CREATE SUBSCRIPTION
TO pub_orders_samuel_singer
FOR SamS
```

Subscriptions with a
subscription expression

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:

```
CREATE SUBSCRIPTION
TO pub_orders ( '856' )
FOR SamS
```

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 189](#).

☞ For more information, see [“CREATE SUBSCRIPTION statement” on page 356](#).

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 91.

Contents

Topic:	page
Design overview	142
Creating publications	143
Publication design for Adaptive Server Enterprise	147
Partitioning tables that do not contain the subscription column	149
Sharing rows among several subscriptions	157
Managing conflicts	165
Ensuring unique primary keys	175
Creating subscriptions	181

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.

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 53.

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:

```
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:

```
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:

```
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:

```
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 **NULL**s.
 - 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

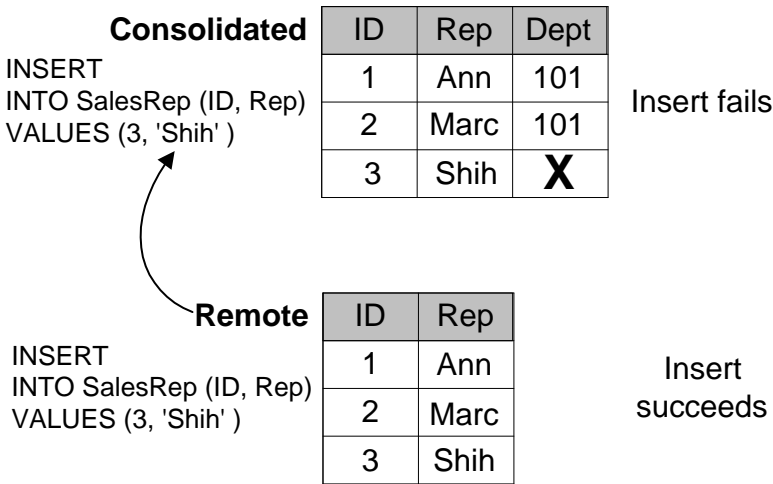
All columns in the primary key must be included in the article.

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.



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 144](#).

Partitioning tables that do not contain the subscription column

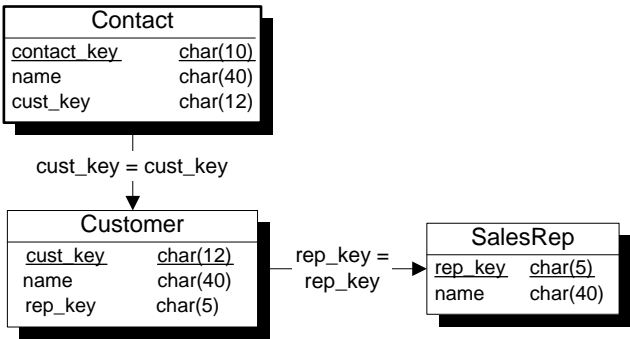
In many cases, the rows of a table need to be partitioned even when the subscription column does not exist in the table. This section describes how to handle this case, using an example.

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:

Table	Description
SalesRep	<p>All sales representatives that work for the company. The SalesRep table has the following columns:</p> <ul style="list-style-type: none"> ♦ rep_key An identifier for each sales representative. This is the primary key. ♦ name The name of each sales representative. <p>The SQL statement creating this table is as follows:</p> <pre>CREATE TABLE SalesRep (rep_key CHAR(12) NOT NULL, name CHAR(40) NOT NULL, PRIMARY KEY (rep_key)) go</pre>
Customer	<p>All customers that do business with the company. The Customer table includes the following columns:</p> <ul style="list-style-type: none"> ♦ 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. <p>The SQL statement creating this table is as follows:</p> <pre>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</pre>

Table	Description
Contact	<p>All individual contacts that do business with the company. Each contact belongs to a single customer. The Contact table includes the following columns:</p> <ul style="list-style-type: none"> ♦ 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. <p>The SQL statement creating this table is:</p> <pre>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)) go</pre>

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 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:

```
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

Add a subscription-list column

The **Contact** table must also be partitioned among the sales representatives, but contains no reference to the sales representative **rep_key** value.

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 157](#).

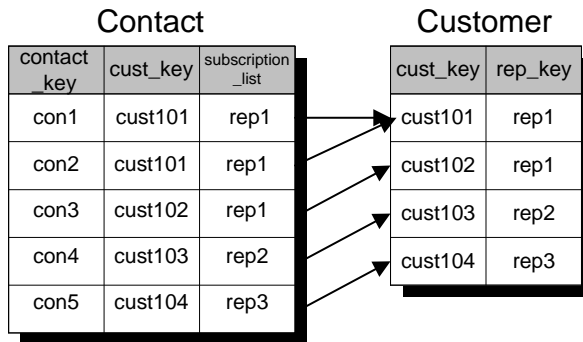
Contact table definition

In the case of the Contact table, the table definition would be changed to the following:

```
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.



☞ 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 105](#).

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.

An INSERT trigger for the Contact table

The trigger for an INSERT on the Contact table sets the **subscription_list** value to the corresponding **rep_key** value from the Customer table:

```
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
```

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
```

Tuning extraction performance

When extracting or synchronizing a user, the *subscription-list* column can cause performance problems as it necessitates a full table scan.

If you are extracting databases for many users, and performance is a problem for you, you can use a **subscription view** to improve performance. The view must contain a subquery, which is used for extraction and synchronization only, and is ignored during log scanning. The tables involved still need to

have triggers defined to maintain the *subscription-list* column.

❖ **To create a subscription view**

1. Design a query that uses a subquery to select the proper rows for a subscription from a table.

For example, continuing the example from the preceding sections, the following query selects the rows of the Contact table for a user subscribed by rep_key value **rep5**:

```
SELECT *
FROM Contact
WHERE 'rep5' = (SELECT rep_key
                FROM Customer
                WHERE cust_key = Contact.cust_key )
```

2. Create a view that contains this subquery. For example:

```
CREATE VIEW Contact_sub_view AS
SELECT *
FROM dbo.Contact
WHERE 'rep5' = ( SELECT rep_key
                FROM dbo.Customer
                WHERE cust_key = dbo.Contact.cust_key )
```

In this view definition, it does not matter what value you use on the left-hand side of the WHERE clause (rep5 in the example above). The replication tools use the subquery for extraction and synchronization only. Rows for which the SUBSCRIBE BY value is equal to the subquery result set are extracted or synchronized.

3. Give the name of the view as a parameter to sp_add_article or sp_modify_article:

```
exec sp_add_remote_table 'Contact'
go
exec sp_add_article SalesRepData,
                    'Contact',
                    NULL,
                    'subscription_list',
                    'Contact_sub_view'
```

The subscription_list column is used for log scanning and the subquery is used for extraction and synchronization.

☞ For more information, see [“Tuning extraction performance for shared rows” on page 162](#), [“sp_add_article procedure” on page 379](#), and [“sp_modify_article procedure” on page 396](#).

Sharing rows among several subscriptions

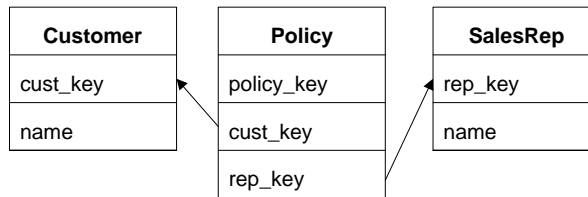
There are cases where a row may need to be included in several subscriptions. For example, if instead of the many-to-one relationship between customers and sales representatives that we had above, we may have a many-to-many relationship.

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.



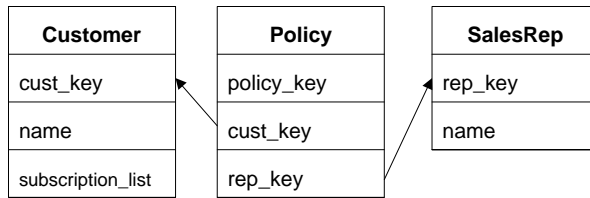
The Policy table has a row for each of a set of policies. Each policy is drawn up for a customer by a particular sales representative. There is a many-to-many relationship between customers and sales representatives, and there may be several policies drawn up between a particular rep/customer pair.

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:



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:

```

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:

```

//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
```

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 rep list 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.

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 1=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 1=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.

❖ To customize the extraction procedure to omit certain triggers

1. Ensure the **quoted_identifier** option is set to ON:

```
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:

```
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
```

Tuning extraction performance for shared rows

When extracting or synchronizing a user, the *subscription-list* column can cause performance problems as it necessitates a full table scan.

If you are extracting databases for many users, and performance is a problem for you, you can use a **subscription view** to improve performance. The view must contain a subquery, which is used for extraction and synchronization only, and is ignored during log scanning. The tables involved still need to have triggers defined to maintain the *subscription-list* column.

❖ To create a subscription view

1. Design a query that uses a subquery to select the proper rows for a subscription from a table.

For example, continuing the example from the preceding sections, the following query selects the rows of the Contact table for a user subscribed by rep_key value **rep5**:

```
SELECT *
FROM Contact
WHERE 'rep5' = (SELECT rep_key
                FROM Customer
                WHERE cust_key = Contact.cust_key )
```

2. Create a view that contains this subquery. For example:

```
CREATE VIEW Customer_sub_view AS
SELECT *
FROM dbo.Customer
WHERE 'repxx' IN ( SELECT rep_key
                  FROM dbo.Policy
                  WHERE dbo.Policy.cust_key = dbo.Customer.cust_key )
```

In this view definition, it does not matter what value you use on the left-hand side of the WHERE clause (repxx in the example above). The replication tools use the subquery for extraction and synchronization only. Rows for which the SUBSCRIBE BY value is in the subquery result set are extracted or synchronized.

3. Give the name of the view as a parameter to sp_add_article or sp_modify_article:

```
exec sp_add_article SalesRepData,
                  'Customer',
                  NULL,
                  'subscription_list',
                  'Customer_sub_view'
```

The subscription_list column is used for log scanning and the subquery is used for extraction and synchronization.

☞ For more information, see [“Tuning extraction performance” on page 155](#), [“sp_add_article procedure” on page 379](#), and [“sp_modify_article procedure” on page 396](#).

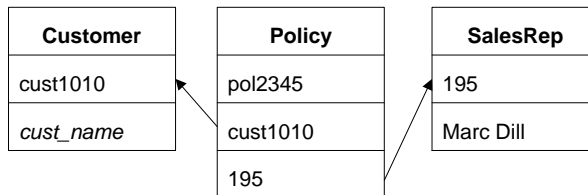
Using the `Subscribe_by_remote` option with many-to-many relationships

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:



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.

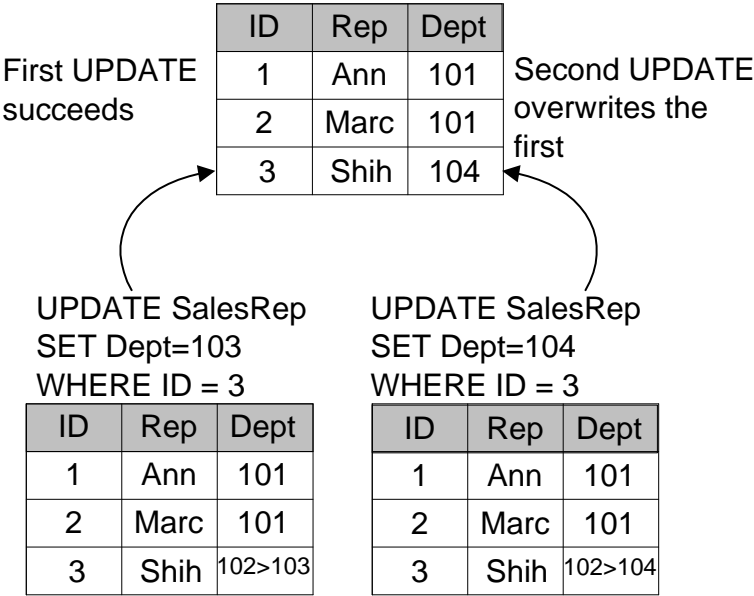
You can use a trigger, which executes after the INSERT, to maintain the subscription-list column.

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.



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 stored procedure to resolve conflicts in a way that makes sense for the data being changed.

Conflicts do not apply to primary keys

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 procedure.

 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 121](#).

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.
- ◆ **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:

```
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

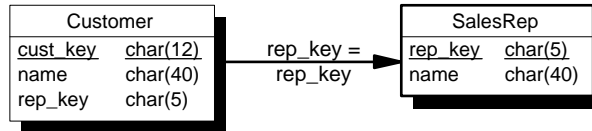
VERIFY_ALL_COLUMNS is set to OFF, if an UPDATE statement updates more than 128 columns, conflict resolution will not work.

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:



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:

```
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:

```
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 Anywhere, 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.

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 = '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:

cust_key	name	rep_key
cust1	C Sports	rep1

The **ConflictLog** table has a single row, showing the conflict:

conflict_key	lost_name	won_name
1	Sea Sports	C Sports

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 168](#).

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:

```
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:

```
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.

```
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 end
    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:

cust_key	name	rep_key
cust1	Consolidated Sports	rep1

The **ConflictLog** table has a single row, showing the conflict and recording the value entered at the remote database:

conflict_key	lost_name	won_name	remote_user
1	Field Sports	Consolidated Sports	field_user

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.

Unreplicated referenced
table errors

Consider the following **SalesRepData** publication:

```
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.

Overview of primary key pools

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),
)
go
```

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	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:

```
CREATE INDEX KeyPoolLocation
ON KeyPool (table_name, location, value)
go
```

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.

```
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.

```
sp_subscription 'create',
    KeyPoolData,
    field_user,
    repl
go
```


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:

```
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
go
```

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:

```
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
go
```

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 procedure 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
    @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 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 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 189](#).

PART III

SQL REMOTE ADMINISTRATION

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

Topic:	page
Deployment overview	186
Test before deployment	187
Synchronizing databases	189
Using the extraction utility	191
Synchronizing data over a message system	198

Deployment overview

	<p>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.</p>
Deployment tasks	<p>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:</p> <ol style="list-style-type: none">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. <p>In the case of large-scale deployments, remote sites are most commonly Adaptive Server Anywhere databases, and this chapter focuses on this case.</p>
Topics covered	<p>This chapter covers the following topics:</p> <ul style="list-style-type: none">◆ 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.

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 some flavor of Windows.

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 machine. Create the Adaptive Server Anywhere databases and load them up with the schema and data on Windows.
- ◆ 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 machines for deployment.
- ◆ Run the extraction utility on Windows, and carry out all database creation and other tasks on the Windows 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 performance tips for Adaptive Server Enterprise users using a *subscription-list* column, see [“Tuning extraction performance” on page 155](#) and [“Tuning extraction performance for shared rows” on page 162](#).

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, if your consolidated database is Adaptive Server Anywhere.
- ◆ As a command-line utility. This is the *dbxtract* utility (Adaptive Server Anywhere), or the *ssxtract* 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 one of the following:
 - ◆ the Sybase Central Create Database wizard (from the Tools menu, choose Adaptive Server Anywhere 9 ► Create Database)
 - ◆ 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 SQL Statements pane runs the *reload.sql* command file:

```
read path\reload.sql
```

where *path* is the path of the reload command file.

When used from Sybase Central, the extraction utility carries out the database unloading task, in the same way that *dbxtract* 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.

- ◆ Create message types for replication.
- ◆ Add a publisher user ID to the database.
- ◆ Add remote users to the database.
- ◆ Add the publication to the database.
- ◆ Create a subscription for the remote users.
- ◆ If you need to specify message link parameters, you must have set them.

☞ For a description of how to carry out these steps, see the tutorial in the chapter [“Tutorials for Adaptive Server Anywhere Users” on page 27](#). For a description of setting message link parameters, see [“Setting message type control parameters” on page 214](#).

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

This section describes how to extract a database for a remote user from the current consolidated database. This section applies only to Adaptive Server Anywhere consolidated databases.

When you complete the Extract Database wizard, it does the following on your machine:

- ◆ Creates the remote database

- ◆ Extracts (unloads) the relevant structures and/or data from the consolidated database to files
- ◆ Loads those files into the newly created remote database

❖ **To extract a database for a remote user (Sybase Central)**

1. In the left pane, select the Adaptive Server Anywhere 9 plug-in.
2. In the right pane, click the Utilities tab.
3. In the right pane, double-click Extract Database.
4. Follow the instructions in the wizard.

Notes

- ◆ You can also access this wizard by clicking Tools ► Adaptive Server Anywhere ► Extract Database.
- ◆ If you use the wizard to extract a non-running database, it is only able to unload the structure and data for you. It cannot create the remote database and reload it. For this reason, we recommend that you always extract from a consolidated database that you are connected to in Sybase Central.
- ◆ You can also invoke the extraction wizard for a particular database or for a particular remote user—Sybase Central automatically fills in the appropriate entries in the wizard.
- ◆ The extraction wizard always extracts (synchronizes) the remote database using the WITH SYNCHRONIZATION option. In those rare cases where you don't want to use this option, you must use the *dbxtract* utility instead.

For more information

For information about the extraction utility options, available as command-line options or as choices presented by the extraction wizard, see [“Extraction utility options” on page 305](#).

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.

☞ For performance tips for Adaptive Server Enterprise users using a *subscription-list* column, see [“Tuning extraction performance” on page 155](#) and [“Tuning extraction performance for shared rows” on page 162](#).

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 305](#)) 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.

An efficient approach to extracting many databases

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.

Extracting groups

If the remote user is a group user ID, the extraction utility extracts all the user IDs of members of that group. You can use this feature to all multiple users on each remote database, using different user IDs, without requiring a custom extraction process.

When a database is extracted for a user, all message link parameters for that user and the groups of which the user is a member are extracted.

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.

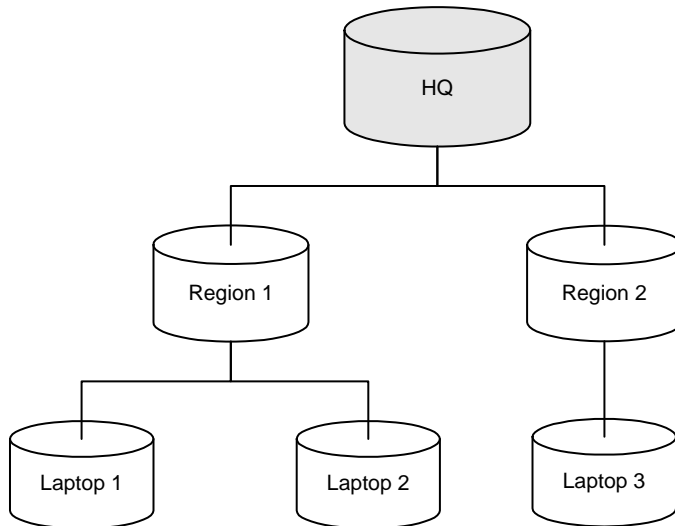
☞ For more information on Adaptive Server Enterprise/Adaptive Server Anywhere issues, see [“Using the extraction utility for Adaptive Server Enterprise” on page 196](#).

- ◆ **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.



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 *ssxtract* include the following:

- ◆ **Grouped procedures** Adaptive Server Anywhere does not support procedure groups, and they are not extracted by *ssxtract*.
- ◆ **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** *ssxtract* 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 Enterprise 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 263 and [“Administering SQL Remote for Adaptive Server Anywhere”](#) on page 241.

Contents

Topic:	page
Management overview	200
Managing SQL Remote permissions	201
Using message types	210
Running the Message Agent	223
Tuning Message Agent performance	228
Encoding and compressing messages	235
The message tracking system	237

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 database can have only one publisher. You can find out who the publisher of an Adaptive Server Anywhere database is at any time in Sybase Central by opening the Users & Groups folder.

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 on an Adaptive Server Anywhere database from Sybase Central. You must connect to the database as a user with full system or database administrator permissions.

❖ To create a new user as the publisher (Sybase Central)

1. In the left pane, select the Users & Groups folder.
2. From the File menu, choose New ► User.
The User Creation wizard appears.
3. Follow the instructions in the wizard. Ensure that the user has a password and is granted Remote DBA authority; this enables the user ID to run the Message Agent.
4. Click Finish to create the user.
5. In the Users & Groups folder, right-click the user you just created and choose Change to Publisher from the popup menu.

❖ To make an existing user the publisher (Sybase Central)

1. In the Users & Groups folder, right-click a user and choose Change to Publisher from the popup menu.

You can also revoke PUBLISH permissions from Sybase Central.

❖ To revoke PUBLISH permissions (Sybase Central)

1. In the Users & Groups folder, right-click the user who has granted PUBLISH permissions and choose Revoke Publisher from 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

- ◆ 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, 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.

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.

Setting REMOTE and 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. Remote users and groups appear in two locations in Sybase Central: in the Users & Groups folder, and in the SQL Remote Users folder. This section applies only to Adaptive Server Anywhere databases.

By default, remote users are created with remote DBA authority. Since the message agent for access to the remote database requires this authority, you shouldn't revoke it.

You cannot create a new remote user until at least one message type is defined in the database.

While you can grant remote permissions to a group, those remote permissions do *not* automatically apply to users in the group (unlike table permissions, for example). To do this, you must explicitly grant remote permissions to each user in the group. Otherwise, remote groups behave exactly like remote users (and are categorized as remote users).

❖ **To add a new user to the database as a remote user (Sybase Central)**

1. In the left pane, select the SQL Remote Users folder.
2. From the File menu, choose New ► SQL Remote User.
The Create a New Remote User wizard appears.
3. Follow the instructions in the wizard.

❖ **To make an existing user remote (Sybase Central)**

1. Open the Users & Groups folder.
2. Right-click the user you want to make remote and choose Change to Remote User from the popup menu.
3. In the resulting dialog, 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.

This user now appears in both the Users & Groups folder and the SQL Remote Users folder.

**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 p.m.

```
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:

```
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:

- ◆ **SEND EVERY** A frequency can be specified in hours, minutes, and seconds in the format 'HH:MM:SS'.

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.

- ◆ **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.

Setting the send frequency in Sybase Central

- ◆ **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.

In Sybase Central, you can specify the send frequency in the following ways:

- ◆ When you make an existing user or group remote. For more information, see [“Granting REMOTE permissions” on page 204](#).
- ◆ On the SQL Remote tab of the property sheet of a remote user or group. You can access the property sheet by right-clicking the remote user or group and choosing Properties from the popup menu.

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:

```
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**, using the file message link:

```
exec sp_grant_consolidate 'hq_user', 'file', address
go
```

Revoking REMOTE and CONSOLIDATE permissions

A user can be removed from a SQL Remote installation by revoking their REMOTE permissions. When you revoke remote permissions from a user or group, you revert that user or group to a normal user/group. You also automatically unsubscribe that user or group from all publications.

Revoking permissions
from Sybase Central

You can revoke REMOTE permissions on Adaptive Server Anywhere databases from Sybase Central.

❖ **To revoke REMOTE permissions (Sybase Central)**

1. Open either the Users & Groups folder or the SQL Remote Users folder.
2. Right-click the remote user or group and choose 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

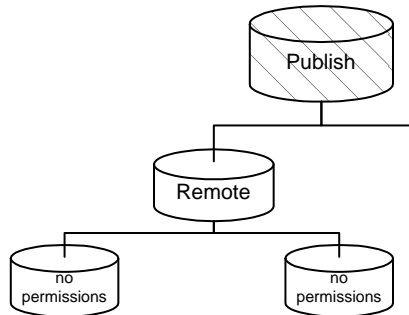
REMOTE permissions can be revoked from a user using the **sp_revoke_remote** procedure. This procedure takes a single argument, which is the user ID of the user. The following statement revokes REMOTE permission from user **S_Beaulieu**.

```
exec sp_revoke_remote 'S_Beaulieu'  
go
```

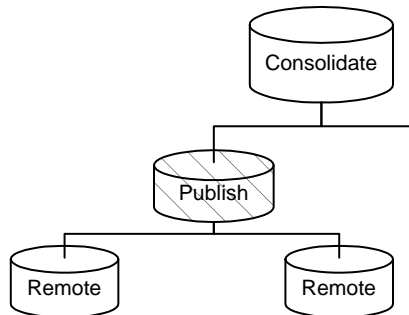
Assigning permissions in multi-tier installations

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.

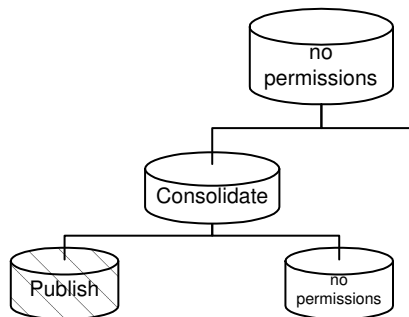
The following picture shows SQL Remote permissions, as granted at the consolidated site of a three-tier installation.



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 operating systems.

☞ For a listing of which message systems are supported on which operating system, see [“Supported Platforms and Message Links” on page 443](#).

For more information

- ◆ For more information on the **file** message system, see [“The file message system” on page 215](#).
- ◆ For more information on the **ftp** message system, see [“The ftp message system” on page 216](#).
- ◆ For more information on the **smtp** message system, see [“The SMTP message system” on page 218](#).
- ◆ For more information on the **mapi** message system, see [“The MAPI message system” on page 220](#).
- ◆ For more information on the **vim** message system, see [“The VIM message system” on page 221](#).

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. Message types appear on the Message Types tab in the right pane when the SQL Remote Users folder is selected. This section applies only to Adaptive Server Anywhere databases.

You must have DBA authority to create and alter message types.

❖ To add a message type (Sybase Central)

1. Connect to a database.
2. In the left pane, open the SQL Remote Users folder for that database.
3. In the right pane, click the Message Types tab.
4. From the File menu, choose New ► Message Type.
The Message Type Creation wizard appears.
5. In the Message Type Creation wizard, enter a message type name. The name should correspond to a message-type DLL already installed in your Adaptive Server Anywhere directory. Click Next.
6. Enter a publisher address and click Finish 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. You cannot change the name of an existing message type; instead, you must delete it and create a new message type with the new name.

❖ To alter a message type (Sybase Central)

1. In the left pane, open the SQL Remote Users folder for a database.
2. In the right pane, click the Message Types tab.
3. In the right pane, right-click the message type you wish to alter and choose Properties from the popup menu.
4. On the property sheet, configure the various options.

If you wish to drop a message type from the installation, you can do so.

❖ To drop a message type (Sybase Central)

1. In the left pane, open the SQL Remote Users folder for a database.
2. In the right pane, click the Message Types tab.
3. In the right pane, right-click the message type you wish to alter and choose Delete from the popup menu.

Creating message types for Windows CE

From within the Sybase Central Utilities tab (select Adaptive Server Anywhere 9 in the left pane, and then click the Utilities tab in the right pane), if you have Windows CE services installed, you have an option to set up SQL Remote for ActiveSync synchronization. This sets your folder for FILE message link messages to be the ActiveSync folder. When you dock your Windows CE machine to your desktop machine, ActiveSync keeps the files in your desktop machine's ActiveSync folder synchronized with those in the Windows CE ActiveSync folder.

Using commands to work with message types

❖ To create a message type (SQL)

1. Make sure you have decided on an address for the publisher under the message type.
2. Execute a CREATE REMOTE MESSAGE TYPE command.

For Adaptive Server Anywhere, the CREATE REMOTE MESSAGE TYPE 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 (SQL)

1. Make sure you have decided on a new address for the publisher under the message type.
2. Execute an ALTER REMOTE MESSAGE TYPE statement.

For Adaptive Server Anywhere, the ALTER REMOTE MESSAGE TYPE 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, *type-name* is one of the message systems supported by SQL Remote, and *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.

❖ To drop a message type (SQL)

1. Execute a DROP REMOTE MESSAGE TYPE statement.

For Adaptive Server Anywhere, the DROP REMOTE MESSAGE TYPE statement has the following syntax:

DROP REMOTE MESSAGE TYPE *type-name*

For Adaptive Server Enterprise, use the **sp_drop_remote_type** procedure in the same way as creating a message type. This procedure takes the following arguments:

sp_drop_remote_type *type-name*

In these statements, *type-name* is one of the message systems supported by SQL Remote.

☞ See also

- ◆ “CREATE REMOTE MESSAGE TYPE statement” on page 355
- ◆ “ALTER REMOTE MESSAGE TYPE statement” on page 353
- ◆ “DROP REMOTE MESSAGE TYPE statement” on page 360

Setting message type control parameters

Each message link has several parameters that govern aspects of its behavior. The parameters differ from message system to message system, but all are managed in the same way.

When you first use the Message Agent for a particular message link, it displays a dialog box showing a set of parameters that control the behavior of the link. These parameters may be a user ID for the message system, a host name where ftp messages are held, and so on. The parameters you enter are saved by the Message Agent. You can also set these parameters explicitly.

Message link parameters stored in the database The message control parameters are held in the database. You can set the options as follows:

❖ To set a message control parameter (Adaptive Server Anywhere)

1. Execute the following statement:

```
SET REMOTE link-name OPTION  
[username.]option-name = option-value
```

❖ To set a message control parameter (Adaptive Server Enterprise)

1. Execute the following statement:

```
exec sp_link_option link-name, [username],  
option-name, option-value
```

You can see the current message link parameters by querying the sys.sysremoteoptions view (Adaptive Server Anywhere) or the sr_remoteoptions view (Adaptive Server Enterprise).

Holding the message link parameters on disk Earlier versions of this software stored the message link parameters outside the database. You can still use this method, but storing the parameters inside the database is recommended unless you have specific reasons to choose otherwise.

The message link control parameters are stored in the following places:

- ◆ **Windows** In the registry, at the following location:

```
\\HKEY_CURRENT_USER
  \Software
    \Sybase
      \SQL Remote
```

The parameters for each message link go in a key under the SQL Remote key, with the name of the message link (4, *smtp*, and so on).

- ◆ **NetWare** You should create a file named *dbremote.ini* in the `sys:\system` directory to hold the FILE system directory setting. This file is not a Windows-format INI file: it must consist of a single line, holding only the directory name.

For example, if the directory is `user:\dbr43`, then the *dbremote.ini* file would contain the following:

```
user:\dbr43
```

- ◆ **UNIX** The FILE system directory setting is held in the `SQLREMOTE` environment variable.

The *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.

When the Message Agent loads a message link, the link uses the settings of the current publisher or, of a setting is not specified, of groups to which the publisher belongs. On Windows, the first time a version of the Message Agent is run that supports storing the message link parameters in the database, it copies the link options from the registry to the database.

The file message system

SQL Remote can be used even if you do not have a message system in place, by using the **file** message system.

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.

When running as an NT service make sure that the account under which the Message Agent is running has permissions to read and write all necessary directories. This is often a problem when accessing network drives.

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.

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.

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.
- ◆ **Unlink_delay** This is the number of seconds to wait before attempting to delete a file if the previous attempt to delete the file failed. If no value is defined for unlink_delay, then the default behavior is to pause for 1 second after the first failed attempt, 2 seconds after the second failed attempt, 3 seconds after the third failed attempt, and 4 seconds after the fourth failed attempt.

On NetWare, you should create a file named *dbremote.ini* in the *sys:\system* directory to hold the directory setting.

The ftp message system

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.

☞ For a list of operating systems for which ftp is supported, see [“Supported operating systems” on page 445](#).

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.ianywhere.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.
- ◆ **active_mode** This is set to either YES or NO, with the default being NO (passive mode).

Troubleshooting ftp problems

Most problems with the FTP message link are network setup issues. This section contains a list of tests you can try to troubleshoot problems.

Set the DEBUG message control parameter Looking over the debug output should indicate whether you are connecting to the FTP server. If you are connecting, it will indicate which FTP commands are failing.

Ping the ftp server If the FTP link is not able to connect to the FTP server, try testing your systems network configuration. If your system has the **ping** command, try typing the following command:

```
ping ftp-server-name
```

You should see output indicating the IP address of the server and the ping (round trip) time to the server. If you can not ping the server than you have a network configuration problem, and you should contact you network administrator.

Check that passive mode works If the FTP link is connecting to the FTP server, but is unable to open a data connection, make sure that an FTP client can use passive mode to transfer data with the server.

Passive mode is the preferred transfer mode and the default for the FTP message link. In passive mode all data transfer connections are initiated by the client, in this case the message link. In Active mode the server initiates all data connections. If your FTP server is sitting behind an incorrectly configured firewall you may not be able to use the default passive transfer mode. In this situation the firewall blocks socket connections to the FTP server on ports other than the FTP control port.

Using an FTP user program that allows you to set the transfer mode between **active** and **passive**, set the transfer mode to passive and try to upload/download a file. If the client you are using cannot transfer the file without using active mode than you should either reconfigure the firewall

and FTP server to allow passive mode transfers or set the active_mode message control parameter to YES. Active mode transfers may not work in all network configurations. For example: if your client is sitting behind an IP masquerading gateway incoming connections may fail depending on you gateway software.

Check permissions and directory structures If the FTP server is connecting and having problems getting directory listings or manipulating files; make sure your permissions are set up correctly and the directories that you need exist.

Log into the FTP server using an FTP program. Change directories to the location stored in the root_directory parameter. If the directories you need do not show up, the root_directory control parameter may be wrong or the directories may not exist.

Test permissions by fetching a file in your message directory and uploading a file to the consolidated database directory. If you get errors your FTP server permissions are set up incorrectly.

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.

☞ For a list of operating systems for which SMTP is supported, see [“Supported operating systems” on page 445](#).

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.

Troubleshooting

If you can not get the SMTP Link to work try connecting to the SMTP/POP3 server from the same machine on which the Message Agent is running using the same account and password. Use an Internet e-mail program that

supports SMTP/POP3 and make sure to disable the program once the SMTP message link is working.

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:

- ◆ **local_host** This is the name of the local computer. It is useful on machines where SQL Remote is unable to determine the local host name. The local host name is needed to initiate a session with any SMTP server. In most network environments, the local host name can be determined automatically and this entry is not needed.
- ◆ **TOP_supported** SQL Remote uses a POP3 command called TOP when enumerating incoming messages. The TOP command may not be supported by all POP servers. Setting this entry to NO will use the RETR command, which is less efficient but will work with all POP servers. The default is YES.
- ◆ **smtp_authenticate** Determines whether the SMTP link authenticates the user. The default value is YES. Set to NO for no SMTP authentication to be carried out.
- ◆ **smtp_userid** The user ID for SMTP authentication. By default this parameter takes the same value as the **pop3_userid** parameter. The **smtp_userid** only needs to be set if the user ID is different to that on the POP server.
- ◆ **smtp_password** The password for SMTP authentication. By default this parameter takes the same value as the **pop3_password** parameter. The **smtp_password** only needs to be set if the user ID is different to that on the POP server.
- ◆ **smtp_host** This is the name of the computer on which the SMTP server is running. It corresponds to the SMTP host field in the SMTP/POP3 login dialog.
- ◆ **pop3_host** This is the name of the computer on which the POP host is running. It is commonly the same as the SMTP host. It corresponds to the POP3 host field in the SMTP/POP3 login dialog.
- ◆ **pop3_userid** This is used to retrieve mail. The POP user ID corresponds to the user ID field in the SMTP/POP3 login dialog. You must obtain a user ID from your POP host administrator.

-
- ◆ **pop3_password** This is used to retrieve mail. It corresponds to the password field in the SMTP/POP3 login dialog. If all of these five fields are set, the login dialog is not displayed.
 - ◆ **Debug** When set to YES, displays all SMTP and POP3 commands and responses. This is useful for troubleshooting SMTP/POP support problems. Default is NO.

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.

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.

☞ For a list of operating systems for which MAPI is supported, see [“Supported operating systems” on page 445](#).

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 mailbox 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 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 VIM message system

The Vendor Independent Messaging system (VIM) is used in Lotus Notes and in some releases of Lotus cc:Mail.

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

☞ For a list of operating systems for which VIM is supported, see [“Supported operating systems” on page 445](#).

The VIM message system uses the following control parameters:

- ◆ **Path** This corresponds to the Path field in the cc:Mail login dialog. It is not applicable to and is ignored under Lotus Notes.
- ◆ **Userid** This corresponds to the User ID field in the cc:Mail login dialog.
- ◆ **Password** This corresponds to the Password field in the cc:Mail login dialog. If all of Path, Userid, and Password are set, the login dialog is not displayed.
- ◆ **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.

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

On Windows operating systems, the Message Agent for Adaptive Server Enterprise is named *ssremote.exe*, and the Message Agent for Adaptive Server Anywhere is named *dbremote.exe*. On UNIX operating systems, the names are *ssremote* and *dbremote*, respectively.

☞ 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 264](#).

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 206](#).

❖ **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** option.

❖ **To run the Message Agent in batch mode**

1. 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** option.

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 249](#) and [“Adaptive Server Enterprise transaction log and backup management” on page 272](#).

- ◆ **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.

Replicating only
backed-up transactions

By default, the Message Agent processes all committed transactions. When the Message Agent is run with the `-u` option, 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* 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` option 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 option. 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 292](#).

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.

Troubleshooting errors at remote sites

There are obvious obstacles for an administrator who has access only to the consolidated site to troubleshoot errors that occur at remote sites. To assist with this task, you can set up SQL Remote so that portions of the output log from remote sites are delivered to the consolidated site and written to a file. This one file contains logging information from some or all sites in the system.

To set up SQL Remote to collect log information, you must configure both the remote and the consolidated sites.

❖ To configure a remote database to send log information to the consolidated database

1. Set a link option to send log information when an error is encountered.

Execute the following command against the remote database:

```
SET REMOTE link-name OPTION  
PUBLIC.OUTPUT_LOG_SEND_ON_ERROR = 'YES'
```

With this option set, any message that starts with the error indicator 'E' causes SQL Remote to send log information to the consolidated site.

☞ For more information, see [“SET REMOTE OPTION statement \[SQL Remote\]” \[ASA SQL Reference, page 560\]](#).

2. Set a link option to limit the amount of information sent to the consolidated site. This step is optional.

Execute the following command against the remote database:

```
SET REMOTE link-name OPTION  
PUBLIC.OUTPUT_LOG_SEND_LIMIT = 'nnn'
```

The value of this option is the number of bytes at the tail of the output log (that is, the most recent entries) which are sent to the consolidated site. You can use *nnnK* to indicate kilobytes. The default setting is '5K'.

If you supply a value that is too large to fit in the maximum message size, SQL Remote overrides the option value and sends only what will fit in the message.

You can also send log information even in the absence of errors by setting the `OUTPUT_LOG_SEND_NOW` option to YES. SQL Remote then sends the output log information on the next poll and resets the option to 'NO' after the log is sent.

❖ To configure a consolidated site to receive log information

1. Use either the `-ro` or the `-rt` Message Agent option.

☞ For more information, see [“The Message Agent” on page 292](#).

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.

Setting the number of worker threads

The number of worker threads is set on the Message Agent command line, using the `-w` option. 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.

Performance benefits from worker threads

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.

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` option.

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 `-m` option is exceeded,

messages are flushed in a least-recently-used fashion.

This option is provided primarily for customers considering a single consolidated database for thousands of remote databases.

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.

Issues to consider

The issues to consider when tuning the message-receiving process are similar to those when tuning the message-sending process.

- ◆ **Regular messages** Your choices dictate how often the Message Agent polls for incoming messages from remote databases.
- ◆ **Resend requests** You can control how many polls to wait until an out-of-order message arrives, before requesting that it be resent.
- ◆ **Processing incoming messages** 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.

☞ For more information on the message sending process, see [“Tuning the message sending process” on page 232](#).

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 `-rd` 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:

```
dbremote -rd 30s
```

Alternatively, you can poll less frequently, as in the following command line, which polls every five minutes:

```
dbremote -rd 5
```

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` 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 preserve 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 *userX.n* so that **user1.5** is the sixth 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 `-sd` Message Agent 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:

- ◆ **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 230](#).

Polling interval

You control the interval to wait between polls for more data from the transaction log to send using the `-sd` 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` 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 ...
```

If you do not specify the `-ru` option, then a default value is picked by the Message Agent, based on the send interval of the users that have requested that data be resent. The elapsed time between receiving a resend request for a user and rescanning the log does not exceed half of the send interval for that user.

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 encoding and compression

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.

Settings for compatibility

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.

Upgrading SQL Remote

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* for Adaptive Server Anywhere and *ssencod.dll* for Adaptive Server Enterprise) 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 COMPRESSION 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 *dbrmt.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 option 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 SYSREMOTEUSER 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 **remoteuser** 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 **remoteuser** 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.

In general, the **log_send** column has the same value as the **log_sent** column. However, if the **log_send** has a value that is greater than **log_sent**, the Message Agent sends messages to the subscriber immediately on its next run.

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 **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 **resend_count** value at the publisher database, and also sets the publisher's **log_sent** value to the value of **confirm_sent**. This resetting of the **log_sent** value causes operations to be resent.

Users cannot reset log_sent

The **log_sent** value cannot be reset by a user, as it is in a system table.

Message identification

Each message is identified by three values:

- ◆ Its **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 **resend_count** value smaller than **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

Topic:	page
Running the Message Agent	242
Error reporting and handling	245
Transaction log and backup management	249
Using passthrough mode	260

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 199](#).

Starting the Message Agent

The Message Agent has a set of options that control its behavior. The only option that is required for the Message Agent to run is the connection parameters option (-c).

☞ For more information on connection parameters, see “Connection parameters” [ASA Database Administration Guide, page 70].

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

Running the Message Agent as a service

If you are running the Message Agent in continuous mode (not batch mode) 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 Windows **service**. A service 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 “Running the Database Server” [ASA Database Administration Guide, page 3].

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 on the Authorities tab of the remote user's property sheet.

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` option sends output to a log file as well.

The Message Agent may print more information in the output log than in the window. 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.

☞ For more information on the log file, see [“The Message Agent” on page 292](#).

Ignoring errors

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

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

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

Implementing error handling procedures

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

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

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

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

☞ For information on setting the `REPLICATION_ERROR` option, see [“SQL Remote options” on page 313](#).

Example: e-mailing notification of errors

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

A stored procedure

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

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

The following stored procedure implements this notification:

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

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

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

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

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

An audit table

An audit table could be defined as follows:

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

The columns have the following meaning:

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

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

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

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

An example of an error

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

```
Received message from "cons" (0-0000000000-0)

SQL statement failed:  (-193) primary key for table
'reptable' is not unique

INSERT INTO cons.reptable(id,text,last_contact)
VALUES (2,'dave','1997/apr/21 16:02:38.325')

COMMIT WORK
```

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

```
primary key for table 'reptable' is not unique (-193)

INSERT INTO cons.reptable(id,text,last_contact) VALUES
(2,'dave','1997/apr/21 16:02:52.605')
```


Transaction log and backup management

The importance of good backup practices

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

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

☞ For information on transaction log mirrors and other backup procedure information, see “Backup and Data Recovery” [ASA Database Administration Guide, page 337].

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.

Because the transaction log continually grows in size, space can become a concern. You can use an event handler on transaction log size to rename the log when it reaches a given size. Then you can use the DELETE_OLD_LOGS option to clean up log files that are no longer needed.

☞ For more information about controlling transaction log size, see the “BACKUP statement” [ASA SQL Reference, page 263].

Setting the transaction log directory

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

Example

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

```
dbremote -c "eng=server_name;uid=DBA;pwd=SQL" e:\archive
```

Log names are not important

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

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

Backup utility options

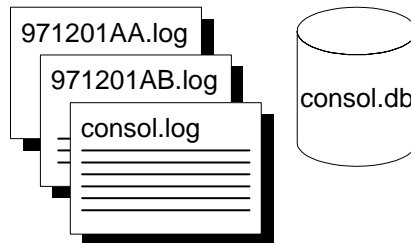
The Adaptive Server Anywhere backup utility has several options, accessible through Sybase Central wizard selections or through *dbbackup* options, that control its behavior.

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* utility, this is the *-r* option.

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



A backup command line

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

```
dbbackup -r -c "uid=DBA;pwd=SQL" c:\archive
```

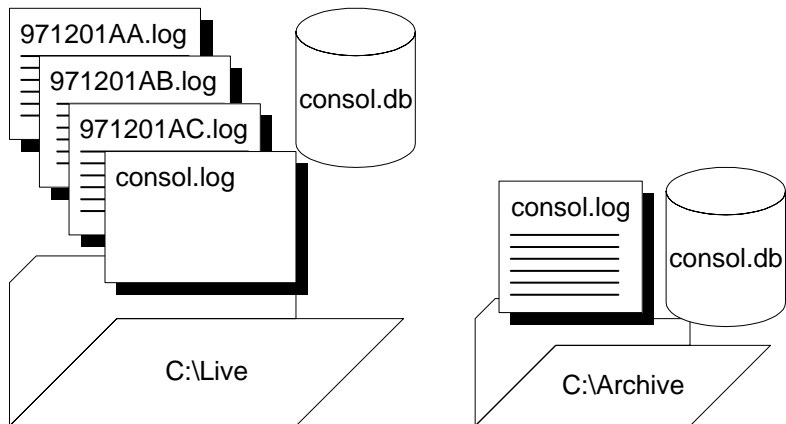
The connection string options would be different for each database.

Effects of the backup

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

1. Backs up the transaction log file, creating a backup file `c:\archive\consol.log`.
2. Renames the existing transaction log file to `971201xx.log`, where `xx` are sequential characters ranging from `AA` to `ZZ`.
3. Starts a new transaction log, as `consol.log`.

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



A Message Agent
command line

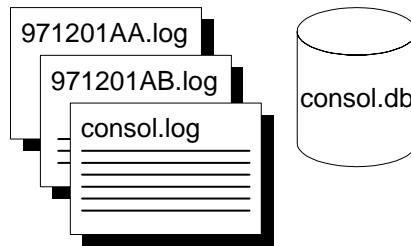
You can run the Message Agent with access to these log files using the following command line:

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

Using the backup directory as the transaction log directory

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

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



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

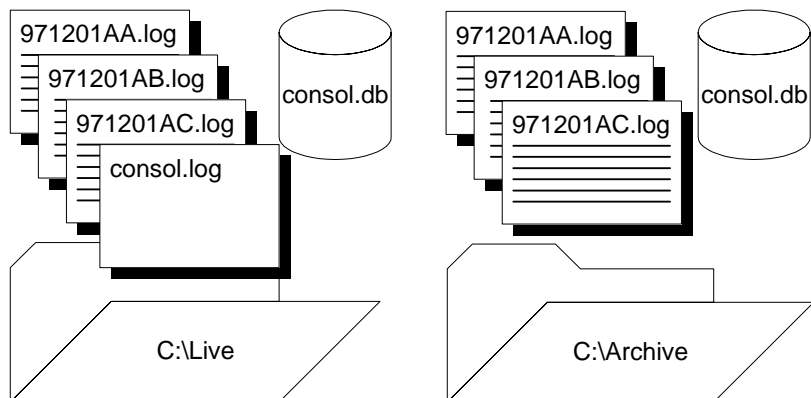
```
dbbackup -r -k -c "uid=DBA;pwd=SQL" c:\archive
```

The connection string options would be different for each database.

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

1. Renames the existing transaction log file to *971201 xx.log*, where *xx* are sequential characters ranging from *AA* to *ZZ*.
2. Backs up the transaction log file to the backup directory, creating a backup file named *971201 xx.log*
3. Starts a new transaction log, as *consol.log*.

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



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

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

Old log names different before 8.01

Prior to release 8.01 of Adaptive Server Anywhere, the old log files were named `yymmdd01.log`, `yymmdd02.log`, and so on. The name change was introduced to allow more old logs to be stored. As the Message Agent scans all the files in the specified directory, regardless of their names, the name change should not affect existing applications.

Managing old transaction logs

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

The replication system no longer needs the logs when all remote databases have received and successfully applied the messages contained in the log files. Remote databases confirm the successful receipt of messages from the consolidated database, and the confirmation sets a value in the consolidated database SQL Remote tables (see [“The message tracking system” on page 237](#)). 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.

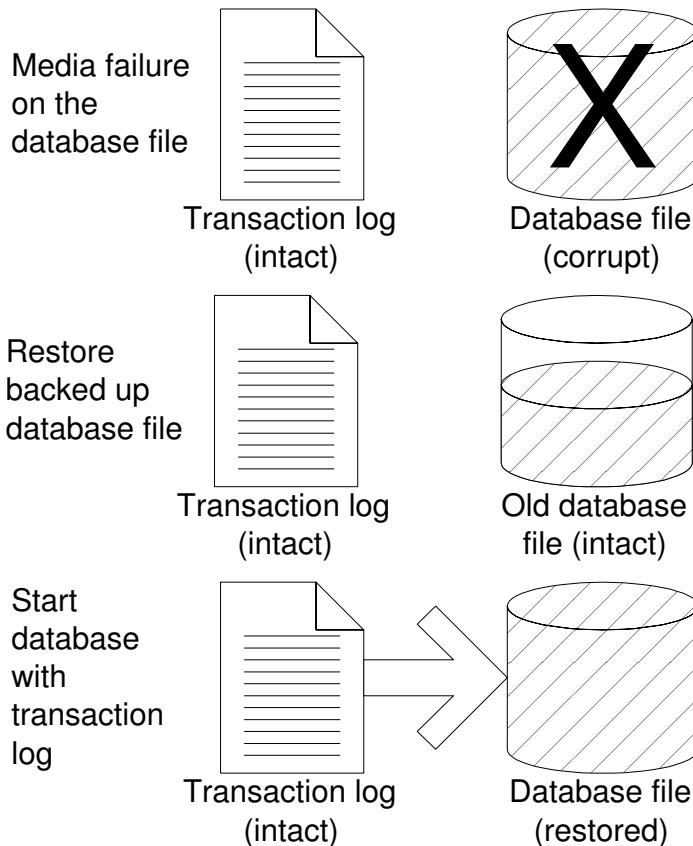
The procedures to follow are easiest to describe if there is only one transaction log file. While this might not be common for consolidated databases, it is described first, followed by a more common but complicated situation with a set of transaction log files.

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` option, to apply the transactions and bring the database file up to date.
4. Start the database in your normal way. Any new activity will be appended to the current transaction log.



Example

This example illustrates recovery using a mirrored transaction log.

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

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

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

```
dbeng9 -a C:\RECOVER\CONSOL.DB
```
7. Shutdown the database server.
8. Backup the recovered database and transaction log from *c:\recover*.
9. Copy the files from *c:\recover* to the appropriate production directories:
 - ◆ Copy *c:\recover\consol.db* to *c:\dbdir\consol.db*
 - ◆ Copy *c:\recover\consol.log* to *c:\dbdir\consol.LOG*, and to *d:\mirdir\consol.mlg*.
10. Restart your system normally.

Recovery with multiple transaction logs

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

❖ To recover the database

1. Make a copy of the database and log file.
2. Restore the database (.db) file, *not* the log file, from tape into a temporary directory.
3. In the temporary directory, start the database, applying the old logs using the -a option, applying the named transaction logs in the correct order.
4. Start the database using the current transaction log and the -a option, to apply the transactions and bring the database file up to date.
5. Start the database in your normal way. Any new activity will be appended to the current transaction log.

Example

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

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

```
dbbackup -c "uid=DBA;pwd=SQL" -r -t E:\BACKDIR
```

This command backs up the transaction log `cons.log` to the directory `e:\backdir`. The transaction log file is then renamed to `datexx.log`, where `date` is the current date and `xx` is the next set of letters in sequence, and a new transaction log is started. The directory `e:\backdir` is then backed up using a third-party utility.

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

```
dbremote -c "uid=DBA;pwd=SQL" C:\DBDIR
```

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

❖ To recover from media failure on the C drive

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

4. Apply the renamed transaction logs in order, as follows

```
dbeng9 -a C:\DBDIR\date00.LOG C:\RECOVER\CONS.DB  
dbeng9 -a C:\DBDIR\date01.LOG C:\RECOVER\CONS.DB
```

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

```
dbeng9 C:\RECOVER\CONS.DB
```
7. Shutdown the database server.
8. Backup the recovered database and transaction log from *c:\recover*.
9. Copy the files from *c:\recover* to the appropriate production directories.
 - ◆ Copy *c:\recover\cons.db* to *c:\dbdir\cons.db*.
 - ◆ Copy *c:\recover\cons.log* to *c:\dbdir\cons.log*, and to *d:\mirdir\cons.mlg*.
10. Restart your system as normal.

Backup procedures at remote databases

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

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

Automatic transaction log renaming You can use the `-x` Message Agent option 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 Upgrade utility” [ASA *Database Administration Guide*, page 542].

Unloading and reloading a database participating in replication

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

☞ For instructions for unloading and reloading a database in “Upgrading the database file format” [*What’s New in SQL Anywhere Studio*, page 178]. The instructions in that section apply to databases involved in SQL Remote replication.

This section describes a manual way of unloading and reloading a database, and is provided in case there are special circumstances that make the use of the more automated procedure referenced above impossible, such as a schema or other significant database change.

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 (manual)**

1. Shut down the existing database.
2. Perform a full off-line backup by copying the database and transaction log files to a secure location.
3. Run the *dbtran* utility to display the starting offset and ending offset of the database's current transaction log file. Note the ending offset for later use.
4. Rename the current transaction log file so that it is not modified during the unload process, and place this file in the off-line directory.
5. Start the existing database.
6. Unload the database.
7. Shut down the existing database. This database and any log file created in this and the previous step is no longer needed.
8. Initialize a new database.
9. Reload the data into the new database.
10. Shut down the new database.
11. Erase the current transaction log file for the new database.
12. Use *dblog* on the new database with the ending offset noted in step 3 as the *-z* option, and also set the relative offset to zero.

```
dblog -x 0 -z 137829 database-name.db
```
13. When you run the Message Agent, provide it with the location of the original off-line directory on its command line.

Using passthrough mode

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

Caution

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

Starting and stopping passthrough

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

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

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

Directing passthrough statements

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

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

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

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

The following statement terminates a passthrough session:

```
PASSTHROUGH STOP ;
```

PASSTHROUGH STOP terminates passthrough mode for all remote users.

Order of application of passthrough statements

Passthrough statements are replicated in sequence with normal replication

Notes on using
passthrough mode

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.

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

Uses and limitations of passthrough mode

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

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

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

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

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

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

- ◆ Add new users.
- ◆ Resynchronize users.
- ◆ Drop users from the setup.

-
- ◆ Change the address, message type, or frequency for a remote user.
 - ◆ Add a column to a table.

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

Passthrough works on only one level of a hierarchy

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

Operations not replicated in passthrough mode

There are special considerations for some statements in passthrough mode.

Calling procedures

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

Control of flow statements and cursor operations

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

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

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

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

However, the following dynamic SQL statement is replicated:

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

Batches

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

CHAPTER 12

Administering SQL Remote for Adaptive Server Enterprise

About this chapter

This chapter details setup and management issues for SQL Remote administrators using Adaptive Server Enterprise as the server for the consolidated database.

Contents

Topic:	page
How the Message Agent for Adaptive Server Enterprise works	264
Running the Message Agent	269
Error reporting and handling	271
Adaptive Server Enterprise transaction log and backup management	272
Making schema changes	275
Using passthrough mode	276

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 223](#).

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 265](#). For more information about how the Message Agent uses the stable queue, see [“Message Agent operation phases” on page 266](#).

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.

Message Agent must be run to reclaim log space

The Message Agent must be run frequently enough to prevent the transaction log from running out of space. The **dump transaction** command does not reclaim space from pages after 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:

- ◆ The database is participating in a Replication Server setup as a primary database, or
- ◆ The database is participating in a Replication Server setup and is using asynchronous procedure calls.

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 277](#).

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.

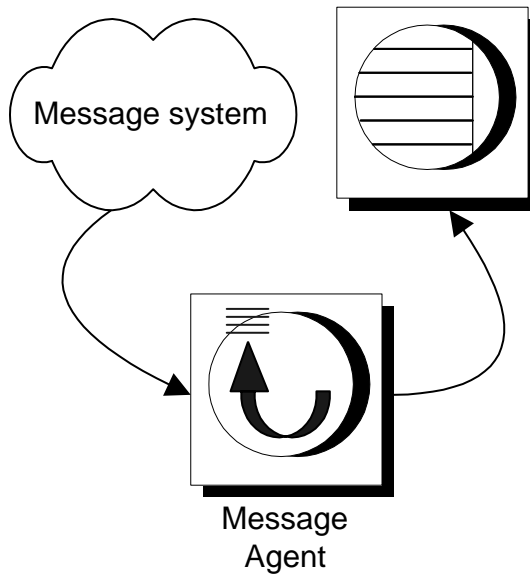
The stable queue consists of a set of tables that contain information on all transactions scanned from the transaction log,

☞ For a description of each of the columns of these tables, see “[Stable Queue tables](#)” on page 348.

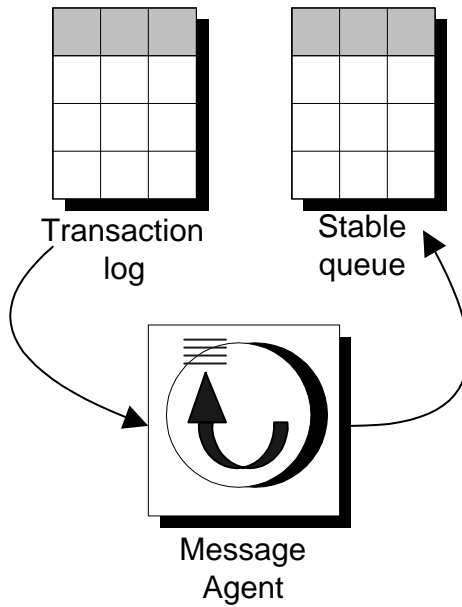
Message Agent operation phases

The Message Agent has the following phases of execution:

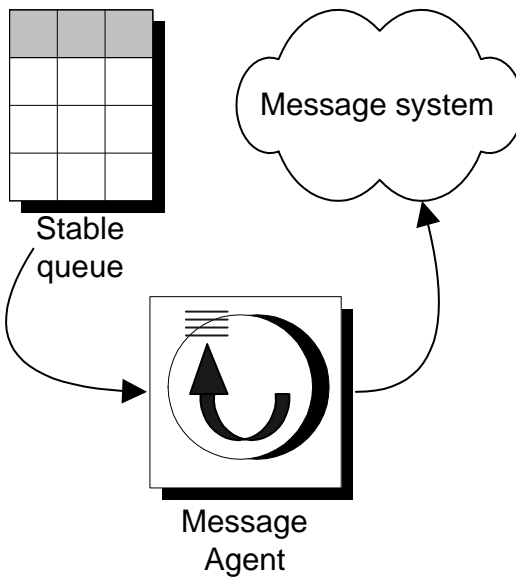
- ◆ **Receiving messages** During this phase, the Message Agent receives incoming messages and applies them to the Adaptive Server Enterprise server.



- ◆ **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 of the database in which the stable queue resides if it is different from the database with SQL Remote system tables.

 For information on running multiple copies of the Message Agent to carry out these tasks, see [“Running multiple Message Agents” on page 269](#).

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 223](#).

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
```

The user for the Message Agent must have insert, update and delete permissions on all replicated tables, in order to apply the changes. Also, the replication error procedure must be created under the Message Agent user ID.

When you setup your Adaptive Server Enterprise database, the scripts *ssremote.sql* and *squeue.sql* must be run under the same user name you use for the Message Agent.

☞ For setup instructions, see [“Setting Up SQL Remote” on page 19](#).

To hide the user password for the Message Agent user ID, you can store the *ssremote* command-line options in a file, and use *ssremote* with the *@filename* parameter. You can use file system security to prevent unauthorized access to the file.

Running multiple Message Agents

The three phases of Message Agent operation are described in the section [“Message Agent operation phases” on page 266](#). 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.

Specifying which phases to execute

The command-line options are as follows:

- ◆ **Receive** The `-r` command-line option 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` option in addition to `-r`.
- ◆ **Scan log** The `-i` command-line option instructs the Message Agent to scan the transaction log into the stable queue while it is running.
- ◆ **Send** The `-s` command-line option 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 one operating system, 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 may print more information in the output log than in the window. 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` or `VARCHAR`. The procedure is called with any error messages and 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 313](#).

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 option 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.

☞ For information on disk mirroring in Adaptive Server Enterprise, see the chapter “Mirroring Database Devices”, in the Adaptive Server Enterprise

System Administration Guide.

- | | |
|---|---|
| Replicating only backed-up transactions | The Message Agent also provides a command-line option (-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 | <p>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.</p> <ul style="list-style-type: none">◆ The Message Agent -u command-line option 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 option 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

```
dbcc settrunc( 'ltm', 'valid' ).
```

The truncation point can be reset with the following command

```
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` options. 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	<p>Passthrough destinations are determined by sp_passthrough_user and sp_passthrough_subscription. Executing either of these procedures determines a set of recipients for any subsequent passthrough statements.</p> <p>Executing either sp_passthrough_user and sp_passthrough_subscription adds to the current list of recipients. The sp_passthrough_stop procedure resets passthrough (that is, resets the list of recipients to be empty).</p> <p>In Adaptive Server Enterprise, sp_passthrough never executes statements in the consolidated database. Passthrough SQL statements are applied only to remote databases.</p>
Passthrough statements	<p>To cause passthrough SQL statements to replicate, you call sp_passthrough.</p> <p>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.</p>
Notes on using passthrough mode	<ul style="list-style-type: none">◆ 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

Topic:	page
When you need to use the SQL Remote Open Server	278
Architecture for Replication Server/SQL Remote installations	279
Setting up SQL Remote Open Server	282
Configuring Replication Server	285
Other issues	287

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 265](#)). 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.

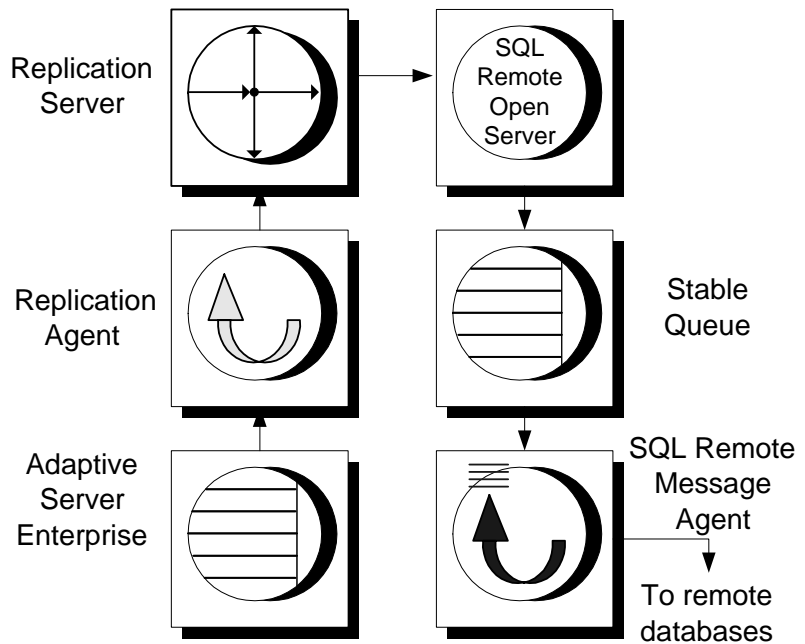
☞ This chapter assumes knowledge of Replication Server. For information, see your Replication Server documentation.

Open Server runtime components required

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.

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 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.



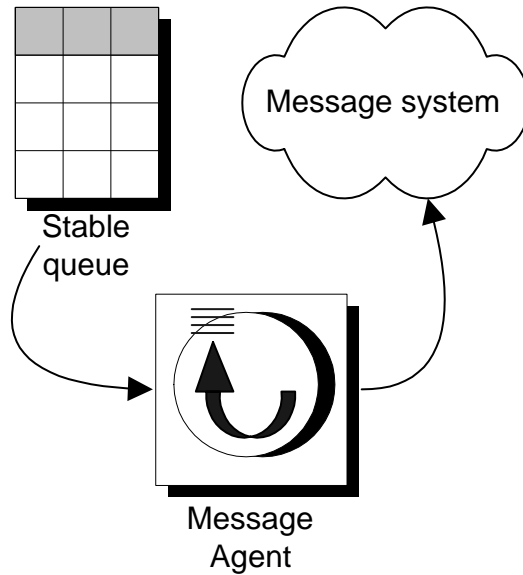
How the pieces fit together

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.

Contents of the stable queue

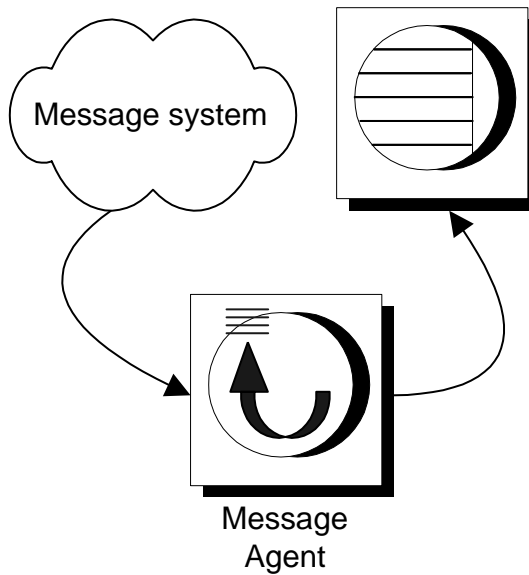
All operations are replicated to the SQL Remote Open Server, which stores them in the stable queue. The stable queue does not have copies of the tables being replicated. It parses the inserts, updates, and deletes to build transactions. All transactions are stored in an image column of a single

table. These transactions are used by the Message Agent to build SQL Remote messages.



Incoming 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.



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 282](#).

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 265](#).

Initial copies of the data

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 nor 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 25](#) 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 19.
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 141.
3. Extract the remote databases. For information on this procedure, see “[Using the extraction utility](#)” on page 191.

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** option, 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 285.

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 10, 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.

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:

```
CONFIGURE CONNECTION TO "ssqueue_server"  
SET dsi_xact_group_size TO '-1'
```

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 78](#)). 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.
Text and image columns	Replication Server does not accept TEXT or IMAGE columns in the primary key of a replication definition. You should include all the columns except for the TEXT and IMAGE columns in the PRIMARY KEY list of your replication definition, and specify all the TEXT and IMAGE columns in the ALWAYS_REPLICATE clause. You should use REPLICATE ALL COLUMNS, instead of REPLICATE MINIMAL COLUMNS in your replication definition. This forces Replication Server to send the pre-image of the TEXT and IMAGE columns to the SQL Remote Open Server whenever an update occurs.
Using the dsi_sql_data_style data style	Replication Server 11.5 has a new dsi_sql_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_sql_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:

```
suspend connection to ssqueue_server
go
resume connection to ssqueue_server
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 options 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.

PART IV

REFERENCE

This part presents reference material for SQL Remote.

CHAPTER 14

Utilities and Options Reference

About this chapter

This chapter provides reference material for the SQL Remote utilities and SQL Remote database options.

It also describes client event-hook stored procedures, which can be used to customize the replication process.

Contents

Topic:	page
The Message Agent	292
The Database Extraction utility	302
The SQL Remote Open Server	310
SQL Remote options	313
SQL Remote event-hook procedures	318

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** } [*options*] [*directory*]

Options

Option	Description
@ <i>filename</i>	Read in options from configuration file
@ <i>envvar</i>	Read in options from environment variable
-a	Do not apply received transactions
-b	Run in batch mode
-c "keyword=value; ..."	Supply database connection parameters
-cq "keyword=value; ..."	Supply database connection parameters for the stable queue (Adaptive Server Enterprise only)
-dl	Display log messages on screen
-ek <i>key</i>	Specify encryption key
-ep	Prompt for encryption key
-e <i>locale-string</i>	Locale setting (Adaptive Server Enterprise only)
-fq	Full scan of the stable queue when sending messages (Adaptive Server Enterprise only)
-g <i>n</i>	Group transactions consisting of less than <i>n</i> operations.
-i	Scan transactions from the transaction log into the stable queue (Adaptive Server Enterprise only).
-k	Close window on completion
-l <i>length</i>	Maximum message length
-m <i>size</i>	Maximum amount of memory used for building messages.
-o <i>file</i>	Output messages to file
-os <i>size</i>	Maximum file size for logging output messages
-ot <i>file</i>	Truncate file and log output messages

Option	Description
-p	Do not purge messages
-q	Run with minimized window
-r	Receive messages
-rd <i>minutes</i>	Polling frequency for incoming messages
-ro <i>filename</i>	Log remote output to file
-rp <i>number</i>	Number of receive polls before message is assumed lost
-rt <i>filename</i>	Truncate, and log remote output to file.
-ru <i>time</i>	Waiting period to re-scan log on receipt of a resend.
-s	Send messages
-sd <i>time</i>	Send polling period
-t	Replicate all triggers (Adaptive Server Anywhere only)
-u	Process only backed up transactions
-ud	On UNIX platforms, run as a daemon.
-v	Verbose operation
-w <i>n</i>	Number of worker threads to apply incoming messages (Not NetWare or Windows CE)
-x [<i>size</i>]	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.

You can also run the Message Agent from your own application by calling

into the DBTools library. For more information, see the file *dbrmt.h* in the *h* subdirectory of your SQL Remote installation directory.

For Adaptive Server Anywhere, the user ID in the Message Agent command 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 224](#).

☞ For information on REMOTE DBA authority, see [“The Message Agent and replication security” on page 243](#).

Option details

@filename Read in options from the supplied file.

The file may contain line breaks, and may contain any set of options. For example, the following command file holds a set of options 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 a command as follows:

```
ssremote @c:\config.txt
```

or

```
dbremote @c:\config.txt
```

@environment-variable Read in options from the supplied environment variable.

The environment variable may contain any set of options. For example, the first of the following pair of statements sets an environment variable holding a set of options 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 -r
-c "eng=myserver;dbn=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 option can help detect problems with incoming messages. Used without `-p`, this option 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:\Program Files\Sybase\SQL Anywhere 9\asademo.db`, connecting with user ID **DBA** and password **SQL**:

```
dbremote -c "uid=DBA;pwd=SQL;dbf=c:\Program Files\Sybase\SQL
Anywhere 9\asademo.db"
```

The Message Agent must be run by a user with REMOTE DBA authority or DBA authority.

☞ For information on REMOTE DBA authority, see [“The Message Agent and replication security” on page 243](#).

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:

Parameter	Description
UID	Login ID
PWD	Password
DBN	(optional) Database name. If this parameter is not supplied, the connection defaults to the default database for the login ID.
ENG	Adaptive Server Enterprise name.

-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 at the command prompt and also in the log file if specified.

Specify encryption key (-ek) This option allows you to specify the encryption key for strongly encrypted databases directly at the command prompt. If you have a strongly encrypted database, you must provide the encryption key to use the database or transaction log in any way, including offline transaction logs. For strongly encrypted databases, you must specify either `-ek` or `-ep`, but not both. The command will fail if you do not specify a key for a strongly encrypted database.

Prompt for encryption key (-ep) This option allows you to specify that you want to be prompted for the encryption key. This option causes a dialog box to appear, in which you enter the encryption key. It provides an extra measure of security by never allowing the encryption key to be seen in clear text. For strongly encrypted databases, you must specify either `-ek` or `-ep`, but not both. The command will fail if you do not specify a key for a strongly encrypted database.

-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 in the `sr_remoteuser` table.

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 `-r`, `-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 269](#).

-k Close window on completion when used together with the `-o` parameter.

-l length Specifies the maximum length of each message to be sent, in bytes. Longer transactions are split into more than one message. The default is 50000 bytes and the minimum length is 10000.

Caution

The maximum message length must be the same at all sites in an installation.

For platforms with restricted memory allocation, the value must be less than the maximum memory allocation of the operating system.

-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` option).

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` option 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.

-os Specifies the maximum file size for logging output messages. The allowed size can be specified as *n* (bytes), *nK* (Kb), or *nM* (Mb). By default there is no limit, and the minimum limit is 10000 bytes.

Before SQL Remote logs output messages to a file, it checks the current file size. If the log message will make the file size exceed the specified size,

SQL Remote renames the output file to *yymmddxx.dbr* (for *dbremote*) and *yymmddxx.ssr* (for *ssremote*) where *xx* are sequential characters ranging from *AA* to *ZZ*, and *yymmdd* represents the current year, month, and date.

If the Message Agent is running in continuous mode for a long time, this option allows you to manually delete old log files and free up disk space.

-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 *-x*, *-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 *-b* option in addition to *-x*.

-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.

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 -rd 30s
```

polls every thirty seconds.

For more information on polling, see [“Tuning incoming message polling” on page 230](#).

-ro This option is for use at consolidated sites. When remote databases are configured to send output log information to the consolidated database, this option writes the information to a file. The option is provided to help administrators troubleshoot errors at remote sites.

☞ For more information, see [“Troubleshooting errors at remote sites” on page 226](#).

-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 230](#).

-rt This option is for use at consolidated sites. It is identical to the **-ro** option except that the file is truncated on startup.

-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 option 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}

-s 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.

-sd time Control the **send delay** which is the time to wait between polls for more transaction log data to send.

-t All trigger actions are replicated. If you do use this option, 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. This option is available for Adaptive Server Anywhere only.

-u Process only transactions that have been backed up. This option prevents the Message Agent from processing transactions since the latest backup. Using this option, 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 **dump database** or **dump transaction** statement are processed.

-ud On UNIX platforms, you can run the Message Agent as a daemon by supplying the **-ud** option.

If you run the Message Agent as a daemon, you must also supply the **-o** or **-ot** option, to log output information.

If you run the Message Agent as a daemon and are using FTP or SMTP message links, you must store the message link parameters in the database,

because the Message Agent does not prompt the user for these options when running as a daemon.

☞ For information on message link parameters, see [“Setting message type control parameters” on page 214](#).

-v Verbose output. This option displays the SQL statements contained in the messages to the screen and, if the `-o` or `-ot` option is used, to a log file.

-w n 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 `-w` option 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 `-w` option 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 database server is shut down. This option is available for Adaptive Server Anywhere only.

If the optional *size* qualifier is supplied, the transaction log is renamed only if it is larger than the specified size. The allowed size can be specified as *n* (in bytes), *nK*, or *nM*. The default is 0.

Message system control parameters

SQL Remote uses several registry settings to control aspects of message link behavior.

The message link control parameters are stored in the following places:

◆ **Windows** In the registry, at the following location:

```
\\HKEY_CURRENT_USER
  \\Software
    \\Sybase
      \\SQL Remote
```

◆ **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 210](#).

The Database Extraction utility

You can access the remote database extraction utility in the following ways:

- ◆ From Sybase Central, for interactive use.
- ◆ From the system command prompt, using the *ssxtract* or *dbxtract* utilities. This is useful for incorporating into batch or command files.
ssxtract 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 305](#)) 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 193](#)).

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.

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 (Sybase Central)**

1. Connect to the database.
2. Right-click the database and choose Extract Database from the popup menu.
3. Follow the instructions in the wizard.

❖ **To extract a remote database from a database file or a running database (Sybase Central)**

1. In the left pane, select the Adaptive Server Anywhere 9 plug-in.
2. In the right pane, click the Utilities tab.
3. In the right pane, double-click Extract Database.
4. Follow the instructions in the wizard.

The extraction utility

Purpose To extract a remote Adaptive Server Anywhere database from a consolidated Adaptive Server Enterprise or Adaptive Server Anywhere database.

Syntax { **ssextract** | **dbxtract** } [*options*] [*directory*] *subscriber*

Option	Description
-an <i>database</i>	Creates a database file with the same settings as the database being unloaded and automatically reloads it.
-ac " <i>keyword=value; ...</i> "	Connect to the database specified in the connect string to do the reload.
-b	Do not start subscriptions
-c " <i>keyword=value; ...</i> "	Supply database connection parameters
-d	Unload data only
-e <i>language,charset</i>	Specify the locale to be used
-f	Extract fully qualified publications
-ii	Internal unload, internal reload
-ix	Internal unload, external reload

Option	Description
-j <i>count</i>	Iteration count for view creation statements
-l <i>level</i>	Perform all extraction operations at specified isolation level
-k	Close window on completion
-n	Extract schema definition only
-o <i>file</i>	Output messages to file
-p <i>character</i>	Escape character
-q	Operate quietly: do not print messages or show windows
-r <i>file</i>	Specify name of generated reload Interactive SQL command file (default “ <i>reload.sql</i> ”)
-u	Unordered data
-v	Verbose messages
-x	Use external table loads
-xf	Exclude foreign keys
-xi	External unload, internal reload
-xp	Exclude stored procedures
-xt	Exclude triggers
-xv	Exclude views
-xx	External unload, external load
-y	Overwrite command file without confirmation
directory	The directory to which the files are written. This is not needed if you use -an or -ac
subscriber	The subscriber for whom the database is to be extracted.

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 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*.

If the remote user is a group, then all the user IDs that are members of that group are extracted. This allows multiple users on a remote database with different user IDs, without requiring a custom extraction process.

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 Extraction utility options, see [“Extraction utility options” on page 305](#).

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 in the command. 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 in the command. 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.

```
ssxtract -c "eng=sample_server;dbn=sademo;
uid=dba;pwd=sql" c:\extract joe_remote
```

If connection parameters are not specified, connection parameters from the `SQLCONNECT` environment variable are used, if set.

- ◆ **ssxtract connection parameters** The following connection parameters are supported:

Parameter	Description
UID	Login ID
PWD	Password
DBN	(optional) Database name. If this parameter is not supplied, the connection defaults to the default database for the login ID.
ENG	Adaptive Server Enterprise name.

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.

Use specified locale (-e) 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).

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.

Internal unload, internal load (-ii) Using this option forces the reload script to use the internal UNLOAD and LOAD TABLE statements rather than the Interactive SQL OUTPUT and INPUT statements to unload and load data, respectively.

This combination of operations is the default behavior.

External operations takes the path of the data files relative to the current

working directory of *dbxtract*, while internal statements take the path relative to the server.

Internal unload, external load (-ix) Using this option forces the reload script to use the internal UNLOAD statement to unload data, and the Interactive SQL INPUT statement to load the data into the new database.

External operations takes the path of the data files relative to the current working directory of *dbxtract*, while internal statements take the path relative to the server.

Iteration count for views (-j) If there are nested views in the consolidated database, this option specifies the maximum number of iterations to use when extracting the views.

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 305](#)) 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. You can specify a different file name with this option.

Output the data unordered (-u) By default the data in each table is ordered by primary key. Unloads are quicker with the -u option, 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.

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.

External unload, internal load (-xi) The default behavior for unloading the database is to use the UNLOAD statement, which is executed by the database server. If you choose an external unload, *dbxtract* uses the OUTPUT statement instead. The OUTPUT statement is executed at the client.

External operations takes the path of the data files relative to the current working directory of *dbxtract*, while internal statements take the path relative to the server.

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.

External unload, external load (-xx) Use the OUTPUT statement to unload the data, and the INPUT statement to load the data into the new database.

The default unload behavior is to use the UNLOAD statement, and the default loading behavior is to use the LOAD TABLE statement. The internal UNLOAD and LOAD TABLE statements are faster than OUTPUT and INPUT.

External operations takes the path of the data files relative to the current working directory of *dbxtract*, while internal statements take the path relative to the server.

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** [*options*] [*open-server-name*]

Options

Option	Description
<i>open-server-name</i>	An open server name, which must be declared in the interfaces file.
-c "keyword=value; ..."	Supply database connection parameters
-cq "keyword=value; ..."	Supply database connection parameters for the stable queue
-dl	Display messages in window
-k	Close window on completion
-o <i>file</i>	Output messages to file
-os <i>file</i>	Maximum file size for logging output messages
-ot <i>file</i>	Truncate file and log output messages
-q	Run with minimized window
-ud	Run as a daemon [UNIX]
-v	Verbose operation

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 operating systems.
- ◆ **ssqueue** UNIX operating systems.

Option details **open-server-name** Replication Server must connect to the SQL Remote Open Server, which therefore must have an open server name. This open

server name is set at the command prompt, and must correspond to a master and query entry in the interfaces file 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 interfaces file is named *sql.ini* on Windows operating systems, and *interfaces* on UNIX.

The default value for the open server name is **SSQueue**.

-c 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:

Parameter	Description
UID	Login ID
PWD	Password
DBN	(optional) Database name. If this parameter is not supplied, the connection defaults to the default database for the login ID.
ENG	Server name.

-cq Specify connection parameters for the stable queue. If not supplied, the values default to the **-c** values.

-dl Display messages in the window or at the command prompt and also in the log file.

-k Close window on completion.

-o Append output to a log file. Default is to send output to the screen.

-os Specifies the maximum file size for logging output messages. The allowed size can be specified as *n* (bytes), *nK* (kb), or *nM* (Mb). By default there is no limit, and the minimum limit is 10000 bytes.

Before SQL Remote logs output messages to a file, it checks the current file size. If the log message will make the filesize exceed the specified size, SQL Remote renames the output file to *yymmddxx.dbr* (for dbremote) and *yymmddxx.ssr* (for ssremote) where *xx* are sequential characters ranging from AA to ZZ, and *yymmdd* represents the current year, month, and date.

If the Message Agent is running in continuous mode for a long time, this option allows you to manually delete old log files and free up disk space.

-ot Truncate the log file and then append output messages to it. Default is to send output to the screen.

-q For Windowing operating systems only, starts the Message Agent with a minimized window.

-ud On UNIX platforms, you can run the SQL Remote Open Server as a daemon by supplying the **-ud** option.

If you run as a daemon, you must also supply the **-o** or **-ot** option, to log output information.

-v Verbose output. This option displays the SQL statements contained in the messages to the screen and, if the **-o** option is used, to a log file.

SQL Remote options

Function Replication options are database options included to provide control over replication behavior.

Adaptive Server
Anywhere Syntax **SET [TEMPORARY] OPTION**
[*userid.* | **PUBLIC.**]*option_name* = [*option_value*]

Adaptive Server
Enterprise syntax:
Parameters **exec sp_remote_option** *option-name*, *option-value*

Argument	Description
<i>option_name</i>	The name of the option being changed.
<i>option-value</i>	A string containing the setting for the option.

Description The following options are available.

OPTION	VALUES	DEFAULT
Blob_threshold	integer, in kb	256
Compression	-1 to 9	6
Delete_old_logs	ON, OFF	OFF
External_remote_options	ON, OFF	OFF
Qualify_owners	ON, OFF	OFF
Quote_all_identifiers	ON, OFF	OFF
Replication_error	<i>procedure-name</i>	NULL
Save_remote_passwords	ON, OFF	ON
SR_Date_Format	<i>date-string</i>	yyyy/mm/dd
SR_Time_Format	<i>time-string</i>	hh:nn:ss.Ssssss
SR_Timestamp_Format	<i>timestamp-string</i>	yyyy/mm/dd hh:nn:ss.Ssssss
Subscribe_by_remote	ON,OFF	ON
Verify_threshold	integer	256
Verify_all_columns	ON,OFF	OFF

These options are used by the Message Agent, and should be set for the user

ID specified in the Message Agent command. 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 the largest blob being replicated.

☞ For information on blob replication and Adaptive Server Enterprise, see [“Replication of blobs” on page 83](#).

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.

External_remote_options This option is used by SQL Remote to indicate whether the message link parameters should be stored in the database (OFF) or externally (ON). By default, the setting is OFF.

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 may be 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.

You can use a table with DEFAULT CURRENT REMOTE USER to record the remote site that caused the error.

Save_remote_passwords option When a password is entered into the message link dialog box on first connection, the parameter values are saved. By default, Save_remote_passwords is ON and the password is saved. If you are storing the message link parameters externally, rather than in the database, you may wish not to save the passwords. You can prevent the passwords from being saved by setting this option to NO.

SR_Date_Format option The Message Agent uses this option when replicating columns that store a date. The option is a string build from the following symbols:

Symbol	Description
yy	Two digit year
yyyy	Four-digit year
mm	Two-digit month
mmm	Character format for month
dd	Two-digit day

Each symbol is substituted with the date being replicated.

If you set the **mm** format symbol in upper case, the corresponding characters

are also upper case.

For the digit formats, the case of the option setting controls padding. If the symbols are the same case (such as DD), the number is padded with zeroes. If the symbols are mixed case (such as Mm), the number is not zero padded.

SR_Time_Format option The Message Agent uses this option when replicating columns that store a time. The option is a string build from the following symbols:

Symbol	Description
hh	Two digit hours (24-hour clock)
nn	Two-digit minutes
mm	Two-digit minutes if following a colon (as in hh:mm)
ss[.s...]	Two-digit seconds plus optional fractions of a second.

Using mixed case in the formatting string suppresses leading zeroes.

SR_Timestamp_Format The Message Agent replicates datetime information using this option. For Adaptive Server Anywhere this is the timestamp, datetime, and smalldatetime data types. For Adaptive Server Enterprise, this is the datetime and smalldatetime data types.

The format strings are taken from the SR_Date_Format and SR_Time_Format settings.

The default setting is the SR_Date_Format setting, followed by the SR_Time_Format setting.

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 118](#), and [“Using the Subscribe_by_remote option with many-to-many](#)

[relationships” on page 164.](#)

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:

```
SET OPTION PUBLIC.Verify_all_columns = 'OFF'
```

- ◆ The following statements set the Verify_all_columns option to OFF in Adaptive Server Enterprise:

```
exec sp_remote_option Verify_all_columns, 'OFF'
go
```

In Adaptive Server Enterprise, replication options are used only by SQL Remote.

SQL Remote event-hook procedures

The following stored procedure names and arguments provide the interface for customizing synchronization at SQL Remote databases.

- Notes
- Unless otherwise stated, the following apply to event-hook procedures:
- ◆ The stored procedures must either have DBA authority (Adaptive Server Anywhere) or dbo authority (Adaptive Server Enterprise).
 - ◆ The procedure must not commit or rollback operations, or perform any action that performs an implicit commit. The actions of the procedure are automatically committed by the calling application.
 - ◆ You can troubleshoot the hooks by turning on the Message Agent verbose mode.
 - ◆ The hooks for *dbremote* and *ssremote* differ only in name.

The #hook_dict table

The #hook_dict table is created immediately before a hook is called using the following CREATE statement:

```
CREATE table #hook_dict(  
  name VARCHAR(128) NOT NULL UNIQUE,  
  value VARCHAR(255) NOT NULL )
```

The Message Agent uses the #hook_dict table to pass values to hook functions; hook functions use the #hook_dict table to pass values back to the Message Agent.

sp_hook_dbremote_begin and sp_hook_ssrmmt_begin

Function

Use this stored procedure to add custom actions at the beginning of the replication process.

Rows in #hook_dict table

Name	Values	Description
send	true or false	Indicates if the process is performing the send phase of replication.
receive	true or false	Indicates if the process is performing the receive phase of replication

Description

If a procedure of this name exists, it is called when the Message Agent starts.

sp_hook_dbremote_end and sp_hook_ssrmmt_end

Function

Use this stored procedure to add custom actions just before the Message

Agent exits.

Rows in #hook_dict table

Name	Values	Description
send	true or false	Indicates if the process is performing the send phase of replication.
receive	true or false	Indicates if the process is performing the receive phase of replication
exit code	integer	A non-zero exit code indicates an error.

Description If a procedure of this name exists, it is called as the last event before the Message Agent shuts down.

sp_hook_dbremote_shutdown and sp_hook_ssrm_shutdown

Function Use this stored procedure to initiate a Message Agent shutdown.

Rows in #hook_dict table

Name	Values	Description
send	true or false	Indicates if the process is performing the send phase of replication.
receive	true or false	Indicates if the process is performing the receive phase of replication
shutdown	true or false	This row is false when the procedure is called. If the procedure updates the row to true the Message Agent is shut down.

Description If a procedure of this name exists, it is called when the Message Agent is neither sending nor receiving messages, and permits a hook-initiated shutdown of the Message Agent.

sp_hook_dbremote_receive_begin and sp_hook_ssrm_receive_begin

Function Use this stored procedure to perform actions before the start of the receive phase of replication.

Rows in #hook_dict None

sp_hook_dbremote_receive_end and sp_hook_ssrm_receive_end

Function Use this stored procedure to perform actions after the end of the receive phase of replication.

Rows in #hook_dict None

sp_hook_dbremote_send_begin and sp_hook_ssrmmt_send_begin

Function Use this stored procedure to perform actions before the start of the send phase of replication.

Rows in #hook_dict None

sp_hook_dbremote_send_end and sp_hook_ssrmmt_send_end

Function Use this stored procedure to perform actions after the end of the send phase of replication.

Rows in #hook_dict None

sp_hook_dbremote_message_sent and sp_hook_ssrmmt_message_sent

Function Use this stored procedure to perform actions after any message is sent.

Rows in #hook_dict

Name	Values
remote user	The message destination

sp_hook_dbremote_message_missing and sp_hook_ssrmmt_message_missing

Function Use this stored procedure to perform actions when the Message Agent has determined that one or more messages is missing from a remote user.

Rows in #hook_dict

Name	Values
remote user	The name of the remote user who will have to resend messages.

sp_hook_dbremote_message_apply_begin and sp_hook_ssrmmt_message_apply_begin

Function Use this stored procedure to perform actions just before the Message Agent applies a set of messages from a user.

Rows in #hook_dict

Name	Values
remote user	The name of the remote user who sent the messages about to be applied.

sp_hook_dbremote_message_apply_end and sp_hook_ssrm_message_apply_end

Function

Use this stored procedure to perform actions just after the Message Agent has applied a set of messages from a user.

Rows in #hook_dict

Name	Values
remote user	The name of the remote user who sent the messages that have been applied.

CHAPTER 15

System Objects for Adaptive Server Anywhere

About this chapter

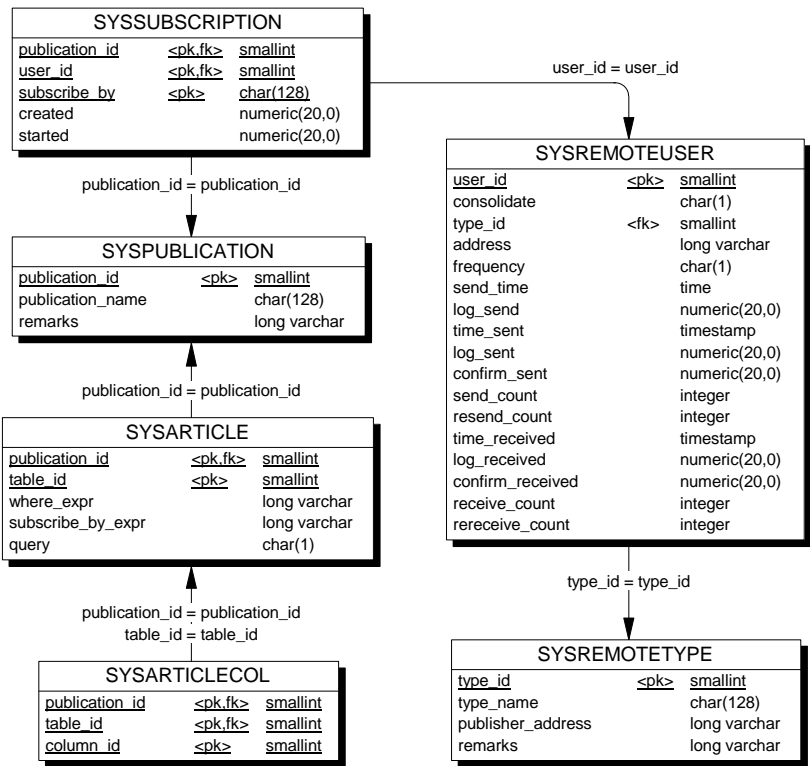
SQL Remote-specific system information is held in the Adaptive Server Anywhere catalog. A more comprehensible version of this information is held in a set of system views.

Contents

Topic:	page
SQL Remote system tables	324
SQL Remote system views	331

SQL Remote system tables

This section describes the system tables used by SQL Remote to define and manage SQL Remote information. In the following diagram, arrows indicate foreign key relations between tables: the arrow leads from the foreign table to the primary table.



These tables are described in more detail in the following sections.

SYSARTICLE table

Function

Each row describes an article in a SQL Remote publication.

Columns

Column	Data type	Description
publication_id	UN-SIGNED INT	The publication of which this article is a part.
table_id	UN-SIGNED INT	Each article consists of columns and rows from a single table. This column contains the table ID for this table.
where_expr	LONG VAR-CHAR	For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.
subscribe_by_expr	LONG VAR-CHAR	For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.
query	CHAR(1)	The SUBSCRIBE BY expression could be a subquery that returns multiple values. This column contains Y or N to indicate if the expression is a subquery (Y) or not (N). This column is used by the Extraction utility

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.

Columns

Column	Data type	Description
publication_id	UN-SIGNED INT	A unique identifier for the publication of which the column is a part.
table_id	UN-SIGNED INT	The table to which the column belongs.
column_id	UN-SIGNED INT	The column identifier, from the SYSCOLUMN system table.

SYSPUBLICATION table

Function Each row describes a SQL Remote publication.

Columns

Column	Data type	Description
publication_id	UNSIGNED INT	A unique identifier for the publication
creator	UNSIGNED INT	The user ID that owns the publication
publication_name	VARCHAR(128)	The name of the publication
remarks	LONG VARCHAR	Comments

SYSREMOTEOPTION table

Function Each row describes the values of a SQL Remote message link parameter.

Columns

Column	Data type	Description
option_id	UNSIGNED INT	An identification number for the message link parameter.
user_id	UNSIGNED INT	The user ID for which the parameter is set.
"setting00"	VARCHAR(255)	The value of the message link parameter.

SYSREMOTEOPTIONTYPE table

Function Each row describes one of the SQL Remote message link parameters.

Columns

Column	Data type	Description
option_id	UN-SIGNED INT	An identification number for the message link parameter.
type_id	SMALLINT	An identification number for the message type that uses this parameter
"option"	VAR-CHAR(128)	The name of the message link parameter.

SYSREMOTETYPE table

Function Each row describes one of the SQL Remote message types, including the publisher address.

Columns

Column	Data type	Description
type_id	SMALLINT	An identification number for the message type.
type_name	VAR-CHAR(128)	The message type. There is a separate row for each of the following: <ul style="list-style-type: none"> ◆ FILE ◆ MAPI ◆ VIM ◆ SMTP
publisher_address	VAR-CHAR(128)	The publisher's address for the message type type_name . SQL Remote receives messages from this address.
remarks	LONG VAR-CHAR	Comments

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.

Columns

Column	Data type	Description
user_id	UNSIGNED INT	The user ID of the user with REMOTE permissions.
consolidate	CHAR(1)	The column contains either an N to indicate a user granted REMOTE permissions, or a Y to indicate a user granted CONSOLIDATE permissions.
type_id	SMALLINT	The ID of the message system used to send messages to this user.
address	LONG VARCHAR	The address to which SQL Remote messages are to be sent. The address must be appropriate for the address_type .
frequency	CHAR(1)	How frequently SQL Remote messages are to be sent. P for Periodically, and A stands for Occasionally.
send_time	TIME	The next time messages are to be sent to this user.
log_send	NUMERIC(20, 0)	If log_send is greater than log_sent , the Message Agent resends messages immediately to the subscriber the next time it is run.
time_sent	TIMESTAMP	The time the most recent message was sent to this subscriber.
log_sent	NUMERIC(20, 0)	The local log offset for the most recently sent operation to this subscriber.
confirm_sent	NUMERIC(20, 0)	The log offset for the most recently confirmed operation from this subscriber.
send_count	INT	The number of SQL Remote messages have been sent to this subscriber.

Column	Data type	Description
resend_count	INT	Counter to ensure messages are applied only once at the subscriber database.
time_received	DATETIME	The time the most recent message was received from this subscriber.
log_received	NUMERIC(20, 0)	The log offset in the subscriber's database for the operation most recently received at the current database.
confirm_received	NUMERIC(20, 0)	The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.
receive_count	INT	How number of messages received from this subscriber.
rereceive_count	INT	Counter to ensure messages are applied only once at the current database.

SYSSUBSCRIPTION table

Function Each row describes a subscription from one user ID (which must have REMOTE permissions) to one publication.

Columns

Column	Data type	Description
publication_id	UN-SIGNED INT	The identifier for the publication to which the user ID is subscribed.
user_id	UN-SIGNED INT	The user ID that is subscribed to the publication.
subscribe_by	VAR-CHAR(128)	For publications with a SUBSCRIBE BY expression, this column holds the matching value for this subscription.
created	NUMERIC(20, 0)	The offset in the transaction log at which the subscription was created.
started	NUMERIC(20, 0)	The offset in the transaction log at which the subscription was started.

SQL Remote system views

This section describes the database views used by SQL Remote to present and summarize SQL Remote information.

SYSARTICLES view

Function Each row lists describes an article.

Columns

Column	Description
publication_name	The publication of which this article is a part.
table_name	Each article consists of columns and rows from a single table. This column contains the name of this table.
where_expr	For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.
subscribe_by_expr	For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.

SYSARTICLECOLS view

Function Each row describes a column that appears in an article.

Columns

Column	Description
publication_name	The name of the publication of which the column is a part.
table_name	The name of the table to which the column belongs.
column_name	The column name.

SYSPUBLICATIONS view

Function Lists the names of all publications.

Columns

Column	Description
publication_name	The name of the publication
createor	The owner of the publication
remarks	Comments

SYSREMOPTIONS view

Function Lists the SQL Remote message link parameters and their values, as stored in the SYSREMOPTION and SYSREMOPTIONTYPE system tables, in more readable form.

Columns

Column	Description
type_name	The message link type.
"option"	The option name.
setting	The option value.

SYSREMOTEUSERS view

Function Lists information about remote users and their status.

Columns

Column	Description
user_name	The user ID of the user with REMOTE permissions.
consolidate	The column contains either an N to indicate a user with REMOTE permissions, or a Y to indicate a user with CONSOLIDATE permissions.
type_name	The name of the message type used to send messages to this user.
address	The address to which SQL Remote messages are to be sent. The address must be appropriate for the address_type .
frequency	How frequently SQL Remote messages are to be sent.
send_time	The next time messages are to be sent to this user.
next_send	The next time messages are to be sent to this user, in a more comprehensible format.

Column	Description
log_send	Messages are sent only to subscribers for whom log_send is greater than log_sent .
time_sent	The time the most recent message was sent to this subscriber.
log_sent	The transaction log offset for the most recently sent operation.
confirm_sent	The transaction log offset for the most recently confirmed operation from this subscriber.
send_count	How many SQL Remote messages have been sent.
resend_count	Counter to ensure messages are applied only once at the subscriber database.
time_received	The time the most recent message was received from this subscriber.
log_received	The log offset in the subscriber's database for the operation most recently received at the current database.
confirm_received	The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.
receive_count	How many messages have been received.
rereceive_count	Counter to ensure messages are applied only once at the current database.

SYSSUBSCRIPTIONS view

Function Each row lists information about a subscription.

Columns

Column	Description
publication_name	The name of the publication to which the user ID is subscribed.
user_name	The user ID that is subscribed to the publication.
subscribe_by	For publications with a SUBSCRIBE BY expression, this column holds the matching value for this subscription.
created	The offset in the transaction log at which the subscription was created.
started	The offset in the transaction log at which the subscription was started.

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

Topic:	page
SQL Remote system tables	336
SQL Remote system views	344
Stable Queue tables	348

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.

Columns

Column	Data type	Description
current_remote_user	VAR-CHAR(128)	Current remote user (from the Message Agent command line).
current_publisher	VAR-CHAR(128)	Current publisher

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 varhcar(128)  
)
```

The table has a single row.

sr_article table

Function Each row describes an article in a SQL Remote publication.

Columns

Column	Data type	Description
publication_id	INT	The publication of which this article is a part.
table_id	INT	Each article consists of columns and rows from a single table. This column contains the table ID for this table.
where_expr	VARCHAR(128)	For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.
subscribe_by_expr	VARCHAR(128)	For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.
subscribe_by_view	VARCHAR(128)	For articles that contain a subset of the rows defined by a view. This column contains the name of the view.

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.

Columns

Column	Data type	Description
publication_id	INT	A unique identifier for the publication of which the column is a part.
table_id	INT	The table to which the column belongs.
column_id	INT	The column identifier, from the SYSCOLUMN system table.

sr_marker table

Function To ensure that messages received by the Message Agent are sent to remote databases in the same session.

Columns

Column	Data type	Description
marker	DATETIME	A time value indicating when the latest messages were applied.

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

Column	Data type	Description
name	VARCHAR(128)	The name of the object.
type	CHAR(1)	One of the following: <ul style="list-style-type: none"> ◆ U User-defined table ◆ V View ◆ P Procedure

sr_option table

Function

Each row describes a replication option used by SQL Remote.

Columns

Column	Data type	Description
option	VARCHAR(128)	The name of the option.
value	VARCHAR(128)	The setting for the option.

Description

☞ For information about available options, see [“SQL Remote options” on page 313](#).

sr_passthrough table

Function

Each row describes a passthrough operation being sent to a user or to

subscribers to a publication.

Columns

Column	Data type	Description
operation	VARCHAR(20)	A passthrough operation, or piece of a passthrough operation, entered using sp_passthrough or sp_passthrough_piece .
value	VAR-CHAR(255)	A subscription column value indicating which users are to receive the operation.
id	INT	A user who is to receive the operation.

sr_publication table

Function

Each row describes a SQL Remote publication.

Columns

Column	Data type	Description
publication_id	INT	An identifier for the publication
publication_name	VAR-CHAR(128)	The name of the publication.

sr_publisher table

Function

The row holds the user ID of the publisher.

Columns

Column	Data type	Description
user_id	INT	The user ID of the publisher.

sr_remotoption table

Function

Each row describes the values of a SQL Remote message link parameter.

Columns

Column	Data type	Description
option_id	INTEGER	An identification number for the message link parameter.
user_id	INTEGER	The user ID for which the parameter is set.
"setting00"	VAR-CHAR(255)	The value of the message link parameter.

sr_remotetype table

Function Each row describes one of the SQL Remote message link parameters.

Columns

Column	Data type	Description
option_id	INTEGER	An identification number for the message link parameter.
type_id	INTEGER	An identification number for the message type that uses this parameter
"option"	VAR-CHAR(128)	The name of the message link parameter.

sr_remotetable table

Function Each row describes a table that is marked for replication using SQL Remote.

Columns

Column	Data type	Description
table_id	INT	The id of the table.
resolve_name	VAR-CHAR(128)	The name of the stored procedure to be executed in the case of conflicts.
old_row_name	VAR-CHAR(128)	The table that holds the old row name.
remote_row_name	VAR-CHAR(128)	The table that holds the remote row name.

sr_remotetype table

Function Each row describes one of the SQL Remote message types, including the

publisher address.

Columns

Column	Data type	Description
type_id	INT	An identification number for the message type.
type_name	VARCHAR(128)	The message type. There is a separate row for each of the following: <ul style="list-style-type: none"> ◆ FILE ◆ MAPI ◆ VIM ◆ SMTP
publisher_address	VARCHAR(128)	The publisher's address for the message type type_name .

sr_remoteuser 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.

Columns

Column	Data type	Description
user_id	INT	The user ID of the user with REMOTE permissions.
consolidate	CHAR(1)	The column contains either an N to indicate a user granted REMOTE permissions, or a Y to indicate a user granted CONSOLIDATE permissions.
type_id	INT	The ID of the message system used to send messages to this user.
address	VARCHAR(128)	The address to which SQL Remote messages are to be sent. The address must be appropriate for the address_type .

Column	Data type	Description
frequency	CHAR(1)	How frequently SQL Remote messages are to be sent.
send_time	DATETIME	The next time messages are to be sent to this user.
log_send	NUMERIC(20, 0)	Messages are sent only to subscribers for whom log_send is greater than log_sent .
time_sent	DATETIME	The time the most recent message was sent to this subscriber.
log_sent	NUMERIC(20, 0)	The log offset for the most recently sent operation.
confirm_sent	NUMERIC(20, 0)	The log offset for the most recently confirmed operation from this subscriber.
send_count	INT	How many SQL Remote messages have been sent.
resend_count	INT	Counter to ensure messages are applied only once at the subscriber database.
time_received	DATETIME	The time the most recent message was received from this subscriber.
log_received	NUMERIC(20, 0)	The log offset in the subscriber's database for the operation most recently received at the current database.
confirm_received	NUMERIC(20, 0)	The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.
receive_count	INT	How many messages have been received from this subscriber.
rereceive_count	INT	Counter to ensure messages are applied only once at the current database.
filler1	CHAR(255)	Reserved

Column	Data type	Description
filler2	CHAR(255)	Reserved
filler3	CHAR(255)	Reserved
filler4	CHAR(255)	Reserved

sr_subscription table

Function Each row describes a subscription from one user ID (which must have REMOTE permissions) to one publication.

Columns

Column	Data type	Description
publication_id	INT	The identifier for the publication to which the user ID is subscribed.
user_id	INT	The user ID that is subscribed to the publication.
subscribe_by	VAR- CHAR(128)	For publications with a SUBSCRIBE BY expression, this column holds the matching value for this subscription.
created	NU- MERIC(20, 0)	The offset in the transaction log at which the subscription was created.
started	NU- MERIC(20, 0)	The offset in the transaction log at which the subscription was started.
operation	VARCHAR(20)	

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.

Columns

Column	Description
publication_name	The publication of which this article is a part.
table_name	Each article consists of columns and rows from a single table. This column contains the name of this table.
where_expr	For articles that contain a subset of rows defined by a WHERE clause, this column contains the search condition.
subscribe_by_expr	For articles that contain a subset of rows defined by a SUBSCRIBE BY expression, this column contains the expression.
subscribe_by_view	For articles that contain a subset of rows defined by a view, this column contains the name of the view.

sr_articlecols view

Function

Each row describes a column that appears in an article.

Columns

Column	Description
publication_name	The name of the publication of which the column is a part.
table_name	The name of the table to which the column belongs.
column_name	The column name.

sr_publications view

Function

Lists the names of all publications.

Columns

Column	Description
publication_name	The name of the publication

sr_remotoptions view

Function

Lists the SQL Remote message link parameters and their values, as stored in the **remotoption** and **remotoptiontype** system tables, in more readable form.

Columns

Column	Description
type_name	The message link type.
"option"	The option name.
setting	The option value.

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

Column	Description
table_name	The name of the table.
resolve_name	The name of the stored procedure to be executed in the case of conflicts.
old_row_name	The table that holds the old row name.
remote_row_name	The table that holds the remote row name.

sr_remotetypes view

Function

Lists the message types, as stored in the **remotetype** system table.

Columns

Column	Description
type_id	An identification number for the message type.
type_name	The message type. There is a separate row for each of the following: <ul style="list-style-type: none">◆ FILE◆ MAPI◆ VIM◆ SMTP
publisher_address	The publisher's address for the message type type_name .

sr_remoteusers view

Function

Lists information about remote users and their status.

Columns

Column	Description
user_name	The user ID of the user with REMOTE permissions.
consolidate	The column contains either an N to indicate a user granted REMOTE permissions, or a Y to indicate a user granted CONSOLIDATE permissions.
type_name	The name of the message system used to send messages to this user.
address	The address to which SQL Remote messages are to be sent. The address must be appropriate for the address_type.
frequency	How frequently SQL Remote messages are to be sent.
send_time	The next time messages are to be sent to this user.
next_send	The next time messages are to be sent to this user, in a more comprehensible format.
log_send	Messages are sent only to subscribers for whom log_send is greater than log_sent.
time_sent	The time the most recent message was sent to this subscriber.

Column	Description
log_sent	The log offset for the most recently sent operation.
confirm_sent	The log offset for the most recently confirmed operation from this subscriber.
send_count	How many SQL Remote messages have been sent.
resend_count	Counter to ensure messages are applied only once at the subscriber database.
time_received	The time the most recent message was received from this subscriber.
log_received	The log offset in the subscriber's database for the operation most recently received at the current database.
confirm_received	The log offset in the subscriber's database for the most recent operation for which a confirmation message has been sent.
receive_count	How many messages have been received.
rereceive_count	Counter to ensure messages are applied only once at the current database.

sr_subscriptions view

Function Each row lists information about a subscription.

Columns

Column	Description
publication_name	The name of the publication to which the user ID is subscribed.
user_name	The user ID that is subscribed to the publication.
subscribe_by	For publications with a SUBSCRIBE BY expression, this column holds the matching value for this subscription.
created	The offset in the transaction log at which the subscription was created.
started	The offset in the transaction log at which the subscription was started.

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

Column	Data type	Description
version	INT	The stable queue version number
page_id	INT	Transaction log page_id of the last entry scanned.
row_id	INT	Transaction log row_id of the last entry scanned.
confirm_offset	NUMERIC(20,0)	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.
commit_offset	NUMERIC(20,0)	The transaction log offset of the most recent transaction completed before the oldest incomplete transaction.
backup_offset	NUMERIC(20,0)	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 option (replicate only backed up transactions).

Column	Data type	Description
marker	DATETIME	<p>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 <code>time_received</code> column in <code>sr_remoteuser</code>. When the transaction log is scanned and the transactions from that message are scanned into the stable queue, it sets the <code>time_received</code> column of <code>sr_queue_state</code>.</p> <p>The purpose of the column is coordination 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 <code>time_received</code> column of <code>sr_queue_state</code>.</p>
confirmed_id	NUMERIC(20,0)	<p>The sending thread deletes rows with <code>confirmed_id</code> less than this value from <code>sr_confirmed_transaction</code>.</p>

sr_transaction table

Function This table has one row for each transaction in the stable queue.

Columns

Column	Description
offset	The transaction log offset of the commit operation for the transaction. This value uniquely identifies each transaction
user_id	<p>The remote user where the transaction originated. This column holds NULL if the transaction did not originate from a remote user.</p> <p>The user_id column is used to ensure that actions are not replicated back to the remote site that entered them.</p>
data	The transaction itself, in an internal representation.

sr_confirmed_transaction table

Function Each row marks the corresponding row in **sr_transaction**.

Columns

Column	Data type	Description
confirmed_id	NUMERIC (20,0)	A unique ID
offset	NUMERIC (20,0)	A copy of an offset used to mark rows in sr_transaction for deletion.

sr_queue_coordinate table

Function A single row, that coordinates the SQL Remote log scanning thread and the sending thread to access the stable queue and related tables.

Columns

Column	Data type	Description
status	CHAR(1)	N if the stable queue has not yet been used by SQL Remote. I and S if the SQL Remote log scanning thread and sending thread have accessed the queue.

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

Topic:	page
ALTER REMOTE MESSAGE TYPE statement	353
CREATE PUBLICATION statement	354
CREATE REMOTE MESSAGE TYPE statement	355
CREATE SUBSCRIPTION statement	356
CREATE TRIGGER statement	357
DROP PUBLICATION statement	359
DROP REMOTE MESSAGE TYPE statement	360
DROP SUBSCRIPTION statement	361
GRANT CONSOLIDATE statement	362
GRANT PUBLISH statement	363
GRANT REMOTE statement	364
GRANT REMOTE DBA statement	365
PASSTHROUGH statement	366
REMOTE RESET statement	367
REVOKE CONSOLIDATE statement	368
REVOKE PUBLISH statement	369
REVOKE REMOTE statement	370
REVOKE REMOTE DBA statement	371
SET REMOTE OPTION statement	372

Topic:	page
START SUBSCRIPTION statement	373
STOP SUBSCRIPTION statement	374
SYNCHRONIZE SUBSCRIPTION statement	375
UPDATE statement	376

ALTER REMOTE MESSAGE TYPE statement

Function Use this statement to change the publisher's message system, or 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*

message-system: **FILE** | **FTP** | **MAPI** | **SMTP** | **VIM**

address: *string*

Parameters

Parameter	Description
<i>message-system</i>	One of the message systems supported by SQL Remote. It must be one of the following values:
<i>address</i>	A string containing a valid address for the specified message system.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also "ALTER REMOTE MESSAGE TYPE statement [SQL Remote]" [ASA *SQL Reference*, page 240]

["CREATE REMOTE MESSAGE TYPE statement" on page 355](#)

["sp_remote_type procedure" on page 425](#)

CREATE PUBLICATION statement

Function	Use this statement to create a publication. In SQL Remote, publications identify replicated data in both consolidated and remote databases.
Syntax	CREATE PUBLICATION [<i>owner</i> .] <i>publication-name</i> (TABLE <i>article-description</i> , ...) <i>owner</i> , <i>publication-name</i> : <i>identifier</i> <i>article-description</i> : <i>table-name</i> [(<i>column-name</i> , ...)] [WHERE <i>search-condition</i>] [SUBSCRIBE BY <i>expression</i>]
See also	“CREATE PUBLICATION statement” [ASA <i>SQL Reference</i> , page 334]

CREATE REMOTE MESSAGE TYPE statement

Function Use this statement to identify a message-link and return address for outgoing messages from a database.

Syntax **CREATE REMOTE MESSAGE TYPE** *message-system*
ADDRESS *address*
message-system: **FILE** | **FTP** | **MAPI** | **SMTP** | **VIM**
address: *string*

Parameters

Parameter	Description
<i>message-system</i>	One of the supported message systems.
<i>address</i>	The address for the specified message system.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also “CREATE REMOTE MESSAGE TYPE statement [SQL Remote]” [ASA SQL Reference, page 337]

[“GRANT PUBLISH statement” on page 363](#)

[“GRANT REMOTE statement” on page 364](#)

[“GRANT CONSOLIDATE statement” on page 362](#)

[“sp_remote_type procedure” on page 425](#)

[“Using message types” on page 210](#)

CREATE SUBSCRIPTION statement

Function Use this statement to create a subscription for a user to a publication.

Syntax **CREATE SUBSCRIPTION**
TO *publication-name* [(*subscription-value*)]
FOR *subscriber-id*

publication-name: identifier

subscription-value, *subscriber-id*: string

subscriber-id: string

Parameters

Parameter	Description
<i>publication-name</i>	The name of the publication to which the user is being subscribed. This may include the owner of the publication.
<i>subscription-value</i>	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.
<i>subscriber-id</i>	The user ID of the subscriber to the publication. This user must have been granted REMOTE permissions.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also “CREATE SUBSCRIPTION statement [SQL Remote]” [ASA SQL Reference, page 347]

 [“DROP SUBSCRIPTION statement” on page 361](#)

 [“GRANT REMOTE statement” on page 364](#)

 [“SYNCHRONIZE SUBSCRIPTION statement” on page 375](#)

 [“START SUBSCRIPTION statement” on page 373](#)

 [“sp_subscription procedure” on page 431](#)

CREATE TRIGGER statement

Function	Use this statement to create a new trigger in the database. One form of trigger is designed specifically for use by SQL Remote.
Syntax	<pre> CREATE TRIGGER <i>trigger-name</i> <i>trigger-time</i> <i>trigger-event</i>, ... [ORDER <i>integer</i>] ON <i>table-name</i> [REFERENCING [OLD AS <i>old-name</i>] [NEW AS <i>new-name</i>]] [REMOTE AS <i>remote-name</i>]] [FOR EACH { ROW STATEMENT }] [WHEN (<i>search-condition</i>)] [IF UPDATE (<i>column-name</i>) THEN [{ AND OR } UPDATE (<i>column-name</i>)] ...] <i>compound-statement</i> [ELSEIF UPDATE (<i>column-name</i>) THEN [{ AND OR } UPDATE (<i>column-name</i>)] ...] <i>compound-statement</i> END IF] <i>trigger-time</i>: BEFORE AFTER RESOLVE <i>trigger-event</i>: DELETE INSERT UPDATE UPDATE OF <i>column-name</i> [, <i>column-name</i>, ...] </pre>
Parameters	<p>trigger-time 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.</p> <p>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.</p> <p>Trigger events Triggers can be fired by one or more of the following events:</p> <ul style="list-style-type: none"> ◆ DELETE Invoked whenever a row of the associated table is deleted. ◆ INSERT Invoked whenever a new row is inserted into the table associated with the trigger. ◆ 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 <i>column-list</i> is modified.

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	“CREATE TRIGGER statement [SQL Remote]” [<i>ASA SQL Reference</i> , page 377] “UPDATE statement” on page 376

DROP PUBLICATION statement

Function	Use this statement to drop a publication. In SQL Remote, publications identify replicated data in both consolidated and remote databases.
Syntax	DROP PUBLICATION [<i>owner.</i>] <i>publication-name</i> <i>owner, publication-name : identifier</i>
See also	“DROP PUBLICATION statement” [ASA SQL Reference, page 413]

DROP REMOTE MESSAGE TYPE statement

Function Use this statement to delete a message type definition from a database.

Syntax **DROP REMOTE MESSAGE TYPE** *message-system*

message-system: **FILE | FTP | MAPI | SMTP | VIM**

Parameters

Parameter	Description
<i>message-system</i>	One of the message systems supported by SQL Remote.

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 “DROP REMOTE MESSAGE TYPE statement [SQL Remote]” [ASA SQL Reference, page 414]

[“CREATE REMOTE MESSAGE TYPE statement” on page 355](#)

[“ALTER REMOTE MESSAGE TYPE statement” on page 353](#)

[“sp_drop_remote_type procedure” on page 386](#)

[“Using message types” on page 210.](#)

DROP SUBSCRIPTION statement

Function Use this statement to drop a subscription for a user from a publication.

Syntax **DROP SUBSCRIPTION TO** *publication-name* [(*subscription-value*)]
FOR *subscriber-id*, ...

subscription-value: string

subscriber-id: string

Parameters

Parameter	Description
<i>publication-name</i>	The name of the publication to which the user is being subscribed. This may include the owner of the publication.
<i>subscription-value</i>	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.
<i>subscriber-id</i>	The user ID of the subscriber to the publication.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also “DROP SUBSCRIPTION statement [SQL Remote]” [ASA SQL Reference, page 419]

[“CREATE SUBSCRIPTION statement” on page 356](#)

[“DROP SUBSCRIPTION statement” on page 361](#)

GRANT CONSOLIDATE statement

Function Use this statement 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' ]
```

message-system: **FILE | FTP | MAPI | SMTP | VIM**

address: *string*

Parameters

Parameter	Description
<i>userid</i>	The user ID for the user to be granted the permission
<i>message-system</i>	One of the message systems supported by SQL Remote.
<i>address</i>	The address for the specified message system.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also

“GRANT CONSOLIDATE statement [SQL Remote]” [ASA *SQL Reference*, page 460]

[“GRANT REMOTE statement” on page 364](#)

[“REVOKE CONSOLIDATE statement” on page 368](#)

[“GRANT PUBLISH statement” on page 363](#)

[“sp_grant_consolidate procedure” on page 388](#)

GRANT PUBLISH statement

Function	Use this statement to identify the publisher of the current database.
Syntax	GRANT PUBLISH TO <i>userid</i>
Permissions	Must have DBA authority.
Side effects	Automatic commit.
See also	<p>“GRANT PUBLISH statement [SQL Remote]” [ASA SQL Reference, page 462]</p> <p>“GRANT REMOTE statement” on page 364</p> <p>“GRANT CONSOLIDATE statement” on page 362</p> <p>“REVOKE PUBLISH statement” on page 369</p> <p>“CREATE PUBLICATION statement” [ASA SQL Reference, page 334]</p> <p>“CREATE SUBSCRIPTION statement” on page 356</p> <p>“sp_publisher procedure” on page 407</p>

GRANT REMOTE statement

Function	Use this statement 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	GRANT REMOTE TO <i>userid</i> , ... TYPE <i>message-system</i> , ... ADDRESS <i>address-string</i> , ... [SEND { EVERY AT } <i>send-time</i>]										
Parameters	<table><tr><th>Parameter</th><th>Description</th></tr><tr><td><i>userid</i></td><td>The user ID for the user to be granted the permission</td></tr><tr><td><i>message-system</i></td><td>One of the message systems supported by SQL Remote. It must be one of the following values:<ul style="list-style-type: none">◆ FILE◆ FTP◆ MAPI◆ SMTP◆ VIM</td></tr><tr><td><i>address-string</i></td><td>A string containing a valid address for the specified message system.</td></tr><tr><td><i>send-time</i></td><td>A string containing a time specification in the form <i>hh:mm</i>.</td></tr></table>	Parameter	Description	<i>userid</i>	The user ID for the user to be granted the permission	<i>message-system</i>	One of the message systems supported by SQL Remote. It must be one of the following values: <ul style="list-style-type: none">◆ FILE◆ FTP◆ MAPI◆ SMTP◆ VIM	<i>address-string</i>	A string containing a valid address for the specified message system.	<i>send-time</i>	A string containing a time specification in the form <i>hh:mm</i> .
Parameter	Description										
<i>userid</i>	The user ID for the user to be granted the permission										
<i>message-system</i>	One of the message systems supported by SQL Remote. It must be one of the following values: <ul style="list-style-type: none">◆ FILE◆ FTP◆ MAPI◆ SMTP◆ VIM										
<i>address-string</i>	A string containing a valid address for the specified message system.										
<i>send-time</i>	A string containing a time specification in the form <i>hh:mm</i> .										
Permissions	Must have DBA authority.										
Side effects	Automatic commit.										
See also	“GRANT REMOTE statement [SQL Remote]” [ASA SQL Reference, page 463] “GRANT CONSOLIDATE statement” on page 362 “REVOKE REMOTE statement” on page 370 “GRANT PUBLISH statement” on page 363 “sp_grant_remote procedure” on page 391 “Granting and revoking REMOTE and CONSOLIDATE permissions” on page 204										

GRANT REMOTE DBA statement

Function	Use this statement to provide DBA privileges to a user ID, but only when connected from the Message Agent.
Syntax	GRANT REMOTE DBA TO <i>userid</i> , ... IDENTIFIED BY <i>password</i>
Permissions	Must have DBA authority.
Side effects	Automatic commit.
See also	“GRANT REMOTE DBA statement [SQL Remote]” [<i>ASA SQL Reference</i> , page 465] “The Message Agent and replication security” on page 243 “REVOKE REMOTE DBA statement” on page 371

PASSTHROUGH statement

Function	Use this statement 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 <i>userid</i> , ...
Syntax 2	PASSTHROUGH [ONLY] FOR SUBSCRIPTION TO [(<i>owner</i>)]. <i>publication-name</i> [(<i>constant</i>)]
Syntax 3	PASSTHROUGH STOP
Permissions	Must have DBA authority.
Side effects	None.
See also	“PASSTHROUGH statement [SQL Remote]” [ASA <i>SQL Reference</i> , page 507] “sp_passthrough procedure” on page 400

REMOTE RESET statement

Function	Use this statement in custom database-extraction procedures to start all subscriptions for a remote user in a single transaction.
Syntax	REMOTE RESET <i>userid</i>
Permissions	Must have DBA authority.
Side effects	No automatic commit is done by this statement.
See also	“REMOTE RESET statement [SQL Remote]” [ASA <i>SQL Reference</i> , page 520] “START SUBSCRIPTION statement” on page 373

REVOKE CONSOLIDATE statement

Function	Use this statement to stop a consolidated database from receiving SQL Remote messages from this database.
Syntax	REVOKE CONSOLIDATE FROM <i>userid</i> , ...
Permissions	Must have DBA authority.
Side effects	Automatic commit. Drops all subscriptions for the user.
See also	“REVOKE CONSOLIDATE statement [SQL Remote]” [<i>ASA SQL Reference</i> , page 532] “GRANT CONSOLIDATE statement” on page 362 “sp_revoke_consolidate procedure” on page 429

REVOKE PUBLISH statement

Function	Use this statement to terminate the identification of the named user ID as the CURRENT publisher.
Syntax	REVOKE PUBLISH FROM <i>userid</i>
Permissions	Must have DBA authority.
Side effects	Automatic commit.
See also	<p>“REVOKE PUBLISH statement [SQL Remote]” [ASA SQL Reference, page 533]</p> <p>“GRANT PUBLISH statement” on page 363</p> <p>“REVOKE REMOTE statement” on page 370</p> <p>“CREATE PUBLICATION statement” [ASA SQL Reference, page 334]</p> <p>“CREATE SUBSCRIPTION statement” on page 356</p> <p>“sp_publisher procedure” on page 407</p>

REVOKE REMOTE statement

Function	Use this statement to stop a user from being able to receive SQL Remote messages from this database.
Syntax	REVOKE REMOTE FROM <i>userid</i> , ...
Permissions	Must have DBA authority.
Side effects	Automatic commit. Drops all subscriptions for the user.
See also	“REVOKE REMOTE statement [SQL Remote]” [<i>ASA SQL Reference</i> , page 535] “sp_revoke_remote procedure” on page 430

REVOKE REMOTE DBA statement

Function	Use this statement to provide DBA privileges to a user ID, but only when connected from the Message Agent.
Syntax 1	REVOKE REMOTE DBA FROM <i>userid</i> , ...
Permissions	Must have DBA authority.
Side effects	Automatic commit.
See also	“REVOKE REMOTE DBA statement [SQL Remote]” [<i>ASA SQL Reference</i> , page 536] “The Message Agent and replication security” on page 243 “GRANT REMOTE DBA statement” on page 365

SET REMOTE OPTION statement

Function	Use this statement to set a message control parameter for a SQL Remote message link.
Syntax	SET REMOTE <i>link-name</i> OPTION [<i>userid.</i> PUBLIC.] <i>link-option-name</i> = <i>link-option-value</i>
Parameters	<i>link-name</i> : file ftp mapi smtp vim <i>link-option-name</i> : <i>file-option</i> <i>ftp-option</i> <i>mapi-option</i> <i>smtp-option</i> <i>vim-option</i> <i>file-option</i> : debug directory unlink_delay <i>ftp-option</i> : active_mode debug host password port root_directory user <i>mapi-option</i> : debug force_download ipm_receive ipm_send profile <i>smtp-option</i> : debug local_host pop3_host pop3_password pop3_userid smtp_host top_supported <i>vim-option</i> : debug password path receive_all send_vim_mail userid <i>link-option-value</i> : <i>string</i>
Permissions	Must have DBA authority. The publisher can set their own options.
Side effects	Automatic commit.
See also	“sp_link_option procedure” on page 394

START SUBSCRIPTION statement

Function Use this statement to start a subscription for a user to a publication.

Syntax **START SUBSCRIPTION**
TO *publication-name* [(*subscription-value*)]
FOR *subscriber-id*, ...

Parameters

Parameter	Description
<i>publication-name</i>	The name of the publication to which the user is being subscribed. This may include the owner of the publication.
<i>subscription-value</i>	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.
<i>subscriber-id</i>	The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also “START SUBSCRIPTION statement [SQL Remote]” [ASA *SQL Reference*, page 571]

“CREATE SUBSCRIPTION statement” on page 356

“REMOTE RESET statement” on page 367

“SYNCHRONIZE SUBSCRIPTION statement” on page 375

“sp_subscription procedure” on page 431

STOP SUBSCRIPTION statement

Function Use this statement to stop a subscription for a user to a publication.

Syntax **STOP SUBSCRIPTION**
TO *publication-name* [(*subscription-value*)]
FOR *subscriber-id*, ...

Parameters

Parameter	Description
<i>publication-name</i>	The name of the publication to which the user is being subscribed. This may include the owner of the publication.
<i>subscription-value</i>	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.
<i>subscriber-id</i>	The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also “STOP SUBSCRIPTION statement [SQL Remote]” [ASA *SQL Reference*, page 579]

 [“CREATE SUBSCRIPTION statement” on page 356](#)
 [“SYNCHRONIZE SUBSCRIPTION statement” on page 375](#)

SYNCHRONIZE SUBSCRIPTION statement

Function Use this statement to synchronize a subscription for a user to a publication.

Syntax **SYNCHRONIZE SUBSCRIPTION**
TO *publication-name* [(*subscription-value*)]
FOR *remote-user*, ...

Parameters

Parameter	Description
<i>publication-name</i>	The name of the publication to which the user is being subscribed. This may include the owner of the publication.
<i>subscription-value</i>	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.
<i>remote-user</i>	The user ID of the subscriber to the publication. This user must have a subscription to the publication.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also “SYNCHRONIZE SUBSCRIPTION statement [SQL Remote]” [ASA SQL Reference, page 581]

[“CREATE SUBSCRIPTION statement” on page 356](#)

[“START SUBSCRIPTION statement” on page 373](#)

UPDATE statement

Function	Use this statement to modify data in the database.
Syntax 1	UPDATE <i>table-list</i> SET <i>column-name</i> = <i>expression</i> , ... [VERIFY (<i>column-name</i> , ...) VALUES (<i>expression</i> , ...)] [WHERE <i>search-condition</i>] [ORDER BY <i>expression</i> [ASC DESC], ...]
Syntax 2	UPDATE <i>table</i> PUBLICATION <i>publication</i> { SUBSCRIBE BY <i>expression</i> OLD SUBSCRIBE BY <i>expression</i> NEW SUBSCRIBE BY <i>expression</i> } WHERE <i>search-condition</i> <i>expression</i> : <i>value</i> <i>subquery</i>
Usage	<p>Syntax 1 and Syntax 2 are applicable only to SQL Remote.</p> <p>Syntax 2 with no OLD and NEW SUBSCRIBE BY expressions must be used in a BEFORE trigger.</p> <p>Syntax 2 with OLD and NEW SUBSCRIBE BY expressions can be used anywhere.</p>
Permissions	Must have UPDATE permission for the columns being modified.
Side effects	None.
See also	“UPDATE statement [SQL Remote]” [ASA <i>SQL Reference</i> , page 599] “CREATE TRIGGER statement” on page 357

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

Topic:	page
sp_add_article procedure	379
sp_add_article_col procedure	381
sp_add_remote_table procedure	382
sp_create_publication procedure	384
sp_drop_publication procedure	385
sp_drop_remote_type procedure	386
sp_drop_sql_remote procedure	387
sp_grant_consolidate procedure	388
sp_grant_remote procedure	391
sp_link_option procedure	394
sp_modify_article procedure	396
sp_modify_remote_table procedure	398
sp_passthrough procedure	400
sp_passthrough_piece procedure	401
sp_passthrough_stop procedure	403
sp_passthrough_subscription procedure	404
sp_passthrough_user procedure	405
sp_populate_sql_anywhere procedure	406
sp_publisher procedure	407
sp_queue_clean procedure	408

Topic:	page
sp_queue_confirmed_delete_old procedure	409
sp_queue_confirmed_transaction procedure	410
sp_queue_delete_old procedure	411
sp_queue_drop procedure	412
sp_queue_dump_database procedure	413
sp_queue_dump_transaction procedure	414
sp_queue_get_state procedure	415
sp_queue_log_transfer_reset 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	425
sp_remove_article procedure	426
sp_remove_article_col procedure	427
sp_remove_remote_table procedure	428
sp_revoke_consolidate procedure	429
sp_revoke_remote procedure	430
sp_subscription procedure	431
sp_subscription_reset procedure	432

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,
subscribe_by_view

Argument	Description
<i>publication_name</i>	The name of the publication to which the article is to be added.
<i>table_name</i>	The table containing the article.
<i>where_expr</i>	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.
<i>subscribe_by_expr</i>	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 <i>table_name</i> . The default value is NULL.
<i>subscribe_by_view</i>	A view defining the columns and rows to be included in the publication. For more information, see “Tuning extraction performance” on page 155 and “Tuning extraction performance for shared rows” on page 162 .

See also [“sp_add_remote_table procedure” on page 382](#)
[“sp_create_publication procedure” on page 384](#)
[“sp_remove_article procedure” on page 426](#)
“CREATE PUBLICATION statement” [ASA SQL Reference, page 334]

Description Running **sp_add_article** adds an article to a publication. The table must be marked for replication using **sp_add_remote_table** 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 **sp_add_article_col**.

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 275](#).

Example

- ◆ The following statement adds the SalesRep table to a publication named SalesRepData:

```
sp_add_article 'SalesRepData', 'SalesRep'  
go
```

sp_add_article_col procedure

Purpose To add a column to an article in a publication.

Syntax **sp_add_article_col** *publication_name*,
table_name,
column_name

Argument	Description
<i>publication_name</i>	The name of the publication to which the article is to be added.
<i>table_name</i>	The table containing the article.
<i>column_name</i>	The column to be added to the article in a publication

See also [“sp_add_article procedure” on page 379](#)

[“sp_remove_article procedure” on page 426](#)


“ALTER PUBLICATION statement” [ASA SQL Reference, page 238]

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 379](#).

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.

 For more information on the requirements for a quiet system, see [“Making schema changes” on page 275](#).

Example ♦ The following statements add the **emp_id** and **emp_lname** columns of the employee table to a publication named **Personnel**:

```
sp_add_article 'Personnel', employee'
sp_add_article_col 'Personnel', 'employee', 'emp_id'
sp_add_article_col 'Personnel', 'employee', 'emp_lname'
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*]

Argument	Description
<i>table_name</i>	The table to be marked for SQL Remote replication.
<i>resolve_procedure</i>	The name of a stored procedure that carries out actions when a conflict occurs.
<i>old_row_name</i>	The name of a table holding the values in the table when a conflict occurs.
<i>remote_row_name</i>	The name of a table holding the values at the remote database when a conflict-causing UPDATE statement was applied.

Authorization You must be a system administrator to execute this procedure.

See also [“sp_modify_remote_table procedure” on page 398](#)
[“sp_remove_remote_table procedure” on page 428](#)
[“Managing conflicts” on page 165.](#)

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 379](#) and the [“sp_add_article_col procedure” on page 381](#).

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*.

Examples

- ◆ The following statement marks the Customer table for replication, using default conflict resolution:

```
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:

```
exec sp_add_remote_table Customer, Customer_Conflict, old_  
Customer, remote_Customer
```

sp_create_publication procedure

sp_drop_publication procedure

Purpose To drop a publication from the database.

Syntax **sp_drop_publication** *publication_name*

Argument	Description
<i>publication_name</i>	The name of the publication to be dropped

See also [“sp_create_publication procedure” on page 384](#)

“DROP PUBLICATION statement” [ASA *SQL Reference*, page 413]

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 **sp_drop_remote_type** *type_name*

Argument	Description
<i>type_name</i>	The message type to drop. This must be a string containing one of the following: <ul style="list-style-type: none">◆ file◆ ftp◆ smtp◆ mapi◆ vim

See also [“sp_remote_type procedure” on page 425](#)

[“DROP REMOTE MESSAGE TYPE statement” on page 360](#)

Description Drops the named message type from the database.

Example ◆ The following statement drops the MAPI message type from the database:

```
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 412
Description	<p>Drops the SQL Remote system objects from the database, so that it can no longer function in a SQL Remote installation.</p> <p>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.</p> <p>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 412.</p>
Example	<p>♦ The following statements remove SQL Remote system objects from a database:</p> <pre>sp_drop_SQL_remote_type go drop procedure sp_drop_SQL_remote go</pre>

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
<i>user_name</i>	The user ID who will be able to receive SQL Remote messages.
<i>type_name</i>	The message type to be used. This must be one of the following: <ul style="list-style-type: none">◆ file◆ ftp◆ smtp◆ mapi◆ vim
<i>address</i>	A string holding the address, according to the specified message type, to which the replication messages should be sent for this user.
<i>frequency</i>	A string containing one of the following: <ul style="list-style-type: none">◆ SEND EVERY Indicates that messages are sent at a frequency specified by <i>send_time</i>.◆ SEND AT Indicates that messages are sent at a time of day specified by <i>send_time</i>.

Argument	Description
<i>send_time</i>	<p>A string containing a time specification with the following meaning:</p> <ul style="list-style-type: none"> ◆ If <i>frequency</i> is SEND EVERY, specifies a length of time between messages. ◆ If <i>frequency</i> is SEND AT, specifies a time of day at which messages will be sent. <p>If no frequency is specified, the Message Agent sends messages, and then stops.</p>

See also

[“sp_grant_remote procedure” on page 391](#)

[“sp_revoke_consolidate procedure” on page 429](#)

[“GRANT CONSOLIDATE statement” on page 362](#)

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.

```
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 **sp_grant_remote** *user_name*,
type_name,
address
[, *frequency*]
[, *send_time*]

Argument	Description
<i>user_name</i>	The user ID who will be able to receive SQL Remote messages.
<i>type_name</i>	The message type to be used. This must be one of the following: <ul style="list-style-type: none"> ◆ file ◆ ftp ◆ smtp ◆ mapi ◆ vim
<i>address</i>	A string holding the address, according to the specified message type, to which the replication messages should be sent for this user.
<i>frequency</i>	A string containing one of the following: <ul style="list-style-type: none"> ◆ SEND EVERY Indicates that messages are sent at a frequency specified by <i>send_time</i>. ◆ SEND AT Indicates that messages are sent at a time of day specified by <i>send_time</i>.

Argument	Description
<i>send_time</i>	<p>An optional string containing a time specification with the following meaning:</p> <ul style="list-style-type: none">◆ If <i>frequency</i> is SEND EVERY, specifies a length of time between messages.◆ If <i>frequency</i> is SEND AT, specifies a time of day at which messages will be sent. <p>If no frequency is specified, the Message Agent sends messages, and then stops.</p>

See also [“sp_revoke_remote procedure” on page 430](#)

[“GRANT REMOTE statement” on page 364](#)

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:

```
exec sp_grant_remote 'SamS',  
    'mapi',  
    'Singer, Samuel',  
    'SEND EVERY',  
    '02:00'  
go
```

sp_link_option procedure

Purpose	To set a message control parameter for a SQL Remote message link.
Syntax	sp_link_option <i>link-name</i> , <i>userid</i> , <i>option-name</i> , <i>option-value</i>
Parameters	<i>link-name</i> : file ftp mapi smtp vim <i>link-option-name</i> : <i>file-option</i> <i>ftp-option</i> <i>mapi-option</i> <i>smtp-option</i> <i>vim-option</i> <i>file-option</i> : debug directory unlink_delay <i>ftp-option</i> : active_mode debug host password port root_directory user <i>mapi-option</i> : debug force_download ipm_receive ipm_send profile <i>smtp-option</i> : debug local_host pop3_host pop3_password pop3_userid smtp_host top_supported <i>vim-option</i> : debug password path receive_all send_vim_mail userid <i>link-option-value</i> : <i>string</i>
Permissions	Must have DBA authority. The publisher can set their own options.
Side effects	Automatic commit.
See also	“SET REMOTE OPTION statement” on page 372
Description	<p>The Message Agent saves message link parameters when the user enters them in the message link dialog box when the message link is first used. In this case, it is not necessary to use this procedure explicitly. This procedure is most useful when preparing a consolidated database for extracting many databases.</p> <p>The option names are case sensitive. The case sensitivity of option values depends on the option: boolean values are case insensitive, while the case</p>

sensitivity of passwords, directory names, and other strings depend on the cases sensitivity of the file system (for directory names), or the database (for user IDs and passwords).

userid If no *userid* is specified, then the current publisher is assumed.

Option values The option values are message-link dependent. For more information, see the following locations:

- ◆ “The file message system” on page 215.
- ◆ “The ftp message system” on page 216.
- ◆ “The MAPI message system” on page 220.
- ◆ “The SMTP message system” on page 218.
- ◆ “The VIM message system” on page 221.

Example

The following statement sets the FTP host to *ftp.mycompany.com* for the ftp link for user myuser:

```
exec sp_link_option ftp, myuser,  
    host, 'ftp.mycompany.com'
```

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*]
[*subscribe_by_view*]

Argument	Description
<i>publication_name</i>	The name of the publication for which the article is to be modified.
<i>table_name</i>	The table containing the article.
<i>where_expr</i>	<p>This optional argument must be a column name or NULL. The publication includes only rows for which the supplied column name is not NULL.</p> <p>The default value is NULL, in which case no rows are excluded from the publication..</p>
<i>subscribe_by_expr</i>	<p>The new subscription expression defining which rows are to be included in the publication for each subscription.</p> <p>The default value is NULL.</p>
<i>subscribe_by_view</i>	<p>A view defining the rows and columns to be included in the publication. The default is NULL.</p> <p>For more information, see “Tuning extraction performance” on page 155 and “Tuning extraction performance for shared rows” on page 162.</p>

See also [“sp_add_article procedure” on page 379](#)
[“sp_remove_article procedure” on page 426](#)

“ALTER PUBLICATION statement” [ASA SQL Reference, page 238]

Description To change the description of an article in a publication. The WHERE expression, the subscription expression, and the subscription view can each 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 275](#).

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, 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*]

Argument	Description
<i>table_name</i>	A table marked for SQL Remote replication.
<i>resolve_procedure</i>	The name of the new stored procedure for carrying out actions when a conflict occurs.
<i>old_row_name</i>	The name of the new table for holding the values in the table when a conflict occurs.
<i>remote_row_name</i>	The name of the new table for holding the values at the remote database when a conflict-causing UPDATE statement was applied.

See also [“sp_add_remote_table procedure” on page 382](#)
[“sp_remove_remote_table procedure” on page 428](#)
[“Managing conflicts” on page 165.](#)

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,  
    old_Cust, remote_Cust  
go
```

sp_passthrough procedure

Purpose	To send a SQL statement in passthrough mode.				
Syntax	sp_passthrough <i>statement</i>				
	<table><tr><th>Argument</th><th>Description</th></tr><tr><td><i>statement</i></td><td>A string containing a statement to be executed in passthrough mode.</td></tr></table>	Argument	Description	<i>statement</i>	A string containing a statement to be executed in passthrough mode.
Argument	Description				
<i>statement</i>	A string containing a statement to be executed in passthrough mode.				
See also	“sp_passthrough_piece procedure” on page 401 “sp_passthrough_stop procedure” on page 403 “sp_passthrough_subscription procedure” on page 404 “sp_passthrough_user procedure” on page 405 “PASSTHROUGH statement” on page 366				
Description	<p>To send passthrough operations. The recipients of the passthrough statement are determined by previous calls to sp_passthrough_user and sp_passthrough_subscription.</p> <p>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.</p> <div><p>Caution</p><p><i>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.</i></p></div>				
Example	<ul style="list-style-type: none">◆ 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*

Argument	Description
<i>string</i>	A piece of a statement to be executed in passthrough mode.

See also [“sp_passthrough procedure” on page 400](#)
[“sp_passthrough_stop procedure” on page 403](#)
[“sp_passthrough_subscription procedure” on page 404](#)
[“sp_passthrough_user procedure” on page 405](#)
[“PASSTHROUGH statement” on page 366](#)

Description The [“sp_passthrough procedure” on page 400](#) 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 '
    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 '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 400 “sp_passthrough_subscription procedure” on page 404 “sp_passthrough_user procedure” on page 405 “PASSTHROUGH statement” on page 366
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. <pre>exec sp_passthrough_stop go</pre>

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

Argument	Description
<i>publication_name</i>	The name of the publication
<i>subscribe_by</i>	The subscription value for recipients to receive passthrough statements.

See also [“sp_passthrough procedure” on page 400](#)
[“sp_passthrough_piece procedure” on page 401](#)
[“sp_passthrough_stop procedure” on page 403](#)
[“sp_passthrough_user procedure” on page 405](#)
[“PASSTHROUGH statement” on page 366](#)

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 405](#).

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 **sp_passthrough_user** *user_name*

Argument	Description
<i>user_name</i>	The user to be added to the list of recipients.

See also [“sp_passthrough procedure” on page 400](#)
[“sp_passthrough_piece procedure” on page 401](#)
[“sp_passthrough_stop procedure” on page 403](#)
[“sp_passthrough_subscription procedure” on page 404](#)
[“PASSTHROUGH statement” on page 366](#)

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 404](#).
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 403](#).

Example ♦ The following statement adds the user field_user to the list of recipients for passthrough statements:

```
sp_passthrough_user 'field_user'  
go
```

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 <i>ssxtract</i> .
Syntax	sp_populate_sql_anywhere
Description	<p>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.</p> <p>This procedure is used by the <i>ssxtract</i> extraction utility. It should not be called directly.</p>

sp_publisher procedure

Purpose To set the publisher of the current database, or to remove the publisher.

Syntax **sp_publisher** [*user_name*]

Argument	Description
<i>user_name</i>	The user ID to be identifies as the publisher for the database.

See also [“Managing SQL Remote permissions” on page 201.](#)

[“GRANT PUBLISH statement” on page 363](#)

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
go
```

♦ The following statement sets the current database to have no publisher:

```
sp_publisher
go
```

sp_queue_clean procedure

Purpose	This procedure is used by the SQL Remote Message Agent, and should not be called directly.
Syntax	sp_queue_clean
Description	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.

sp_queue_confirmed_delete_old procedure

Purpose	This procedure is used by the SQL Remote Message Agent, and should not be called directly.
Syntax	sp_queue_confirmed_delete_old
Description	This procedure is used by the SQL Remote Message Agent, and should not be called directly. It removes from the stable queue any transactions whose offsets are shown in sr_confirmed_transaction .

sp_queue_confirmed_transaction procedure

Purpose	This procedure is used by the SQL Remote Message Agent, and should not be called directly.
Syntax	sp_queue_confirmed_transaction <i>offset</i>
Description	This procedure is used by the SQL Remote Message Agent, and should not be called directly. It adds the supplied offset to sr_confirmed_transaction . SQL Remote removes from the stable queue any transactions whose offsets match this offset.

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 387
Description	<p>Drops the stable queue system objects from the database, so that the database no longer supports a SQL Remote stable queue.</p> <p>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.</p> <p>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 387.</p>
Examples	<ul style="list-style-type: none">◆ The following statements remove the stable queue objects from the database:<pre>sp_queue_drop go drop procedure sp_queue_drop go</pre>

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 414 “Stable queue recovery issues” on page 273
Description	<p>Keeping the stable queue in a separate database complicates backup and recovery, as consistent versions of the two databases have to be recovered.</p> <p>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.</p> <p>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.</p>

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 413 “Stable queue recovery issues” on page 273
Description	<p>Keeping the stable queue in a separate database complicates backup and recovery, as consistent versions of the two databases have to be recovered.</p> <p>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.</p> <p>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.</p>

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_log_transfer_reset procedure

Purpose	This procedure is used by the SQL Remote Message Agent, and should not be called directly.
Syntax	sp_queue_log_transfer_reset
Description	This procedure is used by the SQL Remote Message Agent, and should not be called directly. It resets the page and row IDs to zero in the sr_queue_state table.

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 <i>start_offset</i> , <i>stop_offset</i>
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	<p>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.</p> <p>In a development phase, this procedure can be useful to reset the server.</p>

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 <i>confirm_offset</i>
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 <i>page_id</i> , <i>row_id</i> , <i>commit_offset</i> , <i>backup_offset</i> , <i>marker</i>
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 <i>offset</i> , <i>user_id</i>
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*]

Argument	Description
<i>operation</i>	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.
<i>user_name</i>	The name of the remote user being reset
<i>offset</i>	Not used
<i>confirm</i>	Not used

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 **sp_remote_option** *option_name*,
option_value

Argument	Description
<i>option_name</i>	The name of one of the SQL Remote options
<i>option_value</i>	The value to which the option is set.

See also [“SQL Remote options” on page 313.](#)

Description The SQL Remote options provide control over replication behavior. The following options are available in Adaptive Server Enterprise:

OPTION	VALUES	DEFAULT
Blob_threshold	Integer, in K	256
Compression	-1 to 9	6
Delete_old_logs	ON, OFF	OFF
Qualify_owners	ON, OFF	OFF
Quote_all_identifiers	ON, OFF	OFF
Replication_error	<i>procedure-name</i>	NULL
SR_Date_Format	<i>time-string</i>	hh:nn:ss.Ssssss
SR_Time_Format	<i>date-string</i>	yyyy/mm/dd
SR_Timestamp_Format	<i>timestamp-string</i>	yyyy/mm/dd hh:nn:ss.Ssssss
Subscribe_by_remote	ON,OFF	ON
Verify_threshold	<i>integer</i>	256
Verify_all_columns	ON,OFF	OFF

☞ For a complete description of these options, see [“SQL Remote options” on page 313.](#)

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.

```
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*

Argument	Description
<i>type_name</i>	The message type to create or alter. This must be one of the following: <ul style="list-style-type: none"> ◆ file ◆ ftp ◆ smtp ◆ mapi ◆ vim
<i>publisher_address</i>	The address of the publisher under the specified message type.

See also [“sp_drop_remote_type procedure” on page 386](#)

[“ALTER REMOTE MESSAGE TYPE statement” on page 353](#)

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.

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

Argument	Description
<i>publication_name</i>	The name of the publication from which the article is to be deleted.
<i>table_name</i>	The table containing the article.

See also [“sp_add_article procedure” on page 379](#)

“ALTER PUBLICATION statement” [ASA *SQL Reference*, page 238]

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

Argument	Description
<i>publication_name</i>	The name of the publication to which the article belongs.
<i>article_name</i>	The article from which the column is to be removed.
<i>column_name</i>	The column to be removed from the article.

See also [“sp_add_article_col procedure” on page 381](#)

[“sp_remove_article procedure” on page 426](#)

“ALTER PUBLICATION statement” [ASA SQL Reference, page 238]

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 381](#). Although the [“sp_add_article procedure” on page 379](#), 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*

Argument	Description
<i>table_name</i>	The table to be marked as not available for SQL Remote replication.

See also [“sp_add_remote_table procedure” on page 382](#)

[“sp_modify_remote_table procedure” on page 398](#)

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*

Argument	Description
<i>user_name</i>	The user ID who will no longer be able to act as a consolidated database.

See also [“sp_grant_consolidate procedure” on page 388](#)

Description 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*

Argument	Description
<i>user_name</i>	The user ID who will no longer be able to receive SQL Remote messages.

See also [“sp_grant_remote procedure” on page 391](#)

Description 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*]

Argument	Description
<i>operation</i>	<p>The operation to be performed. This must be one of the following:</p> <ul style="list-style-type: none"> ♦ create To create a subscription to a given publication for a user. ♦ drop To drop a subscription to a given publication for a user. ♦ start To start a subscription to the named publication. ♦ stop To stop a subscription to the named publication. ♦ synchronize To synchronize a subscription to the named publication.
<i>publication_name</i>	The name of the publication to which the subscription refers.
<i>user_name</i>	The user ID who is being subscribed to the publication.
<i>subscribe_by</i>	The subscription value.

See also [“Creating subscriptions” on page 181.](#)

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.

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
```

sp_subscription_reset procedure

Purpose	To reset all SQL Remote information for all remote users.
Syntax	sp_subscription_reset
Description	This procedure resets all the entries in the sr_remote_user and sr_subscription tables to zero or NULL.

PART V

APPENDIX

The appendix provides additional information that is not necessarily required for everyday use of the application.

APPENDIX A

SQL Remote for Adaptive Server Enterprise and Adaptive Server 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	Topic:	page
	Types of difference	436
	Differences in functionality	437
	Differences in approach	438
	Limitations for Enterprise to Enterprise replication	440

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:

- ◆ **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` options. 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 than 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 [“Partitioning tables that do not contain the subscription expression” on page 105](#), and [“Partitioning tables that do not contain the subscription column” on page 149](#).
- ◆ **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 [“Managing conflicts” on page 120](#), and [“Managing conflicts” on page 165](#).
- ◆ **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.

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:

Adaptive Server Enterprise procedure	Corresponding Adaptive Server Anywhere statement
sp_remote_type	CREATE REMOTE MESSAGING TYPE

Adaptive Server Enterprise procedure	Corresponding Adaptive Server Anywhere statement
sp_remote_type	ALTER REMOTE MESSAGE TYPE
sp_drop_remote_type	DROP REMOTE MESSAGE TYPE
sp_grant_remote	GRANT REMOTE
sp_revoke_remote	REVOKE REMOTE
sp_publisher	GRANT PUBLISH
sp_unpublisher	REVOKE PUBLISH
sp_create_publication	CREATE PUBLICATION
sp_add_article	
sp_add_article_col	
sp_add_article	ALTER PUBLICATION
sp_remove_article	
sp_add_article_col	
sp_remove_article_col	
sp_drop_publication	DROP PUBLICATION
sp_subscription 'create'	CREATE SUBSCRIPTION
sp_subscription 'drop'	DROP SUBSCRIPTION
sp_subscription 'start'	START SUBSCRIPTION
sp_subscription 'stop'	STOP SUBSCRIPTION
sp_subscription 'synchronize'	SYNCHRONIZE SUBSCRIPTION
sp_passthrough_user	PASSTHROUGH FOR USERID
sp_passthrough_subscription	PASSTHROUGH FOR SUBSCRIPTION
sp_passthrough_stop	PASSTHROUGH STOP

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 Adaptive Server Enterprise 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 437](#)). 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.

- ◆ **Synchronize subscription** This is not implemented for Adaptive Server Enterprise remote databases.

APPENDIX B

Supported Platforms and Message Links

About this Appendix	This appendix summarizes the platforms and message links that SQL Remote supports.	
Contents	Topic:	page
	Supported message systems	444
	Supported operating systems	445

Supported message systems

SQL Remote exchanges data among databases using an underlying message system. SQL Remote supports the following message systems:

- ◆ **File sharing** A simple system requiring no extra software.
- ◆ **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. For all systems other than the file sharing system, you must have purchased and installed the appropriate message system software for SQL Remote to function over this system. SQL Remote does not include the underlying message system software.

Supported operating systems

SQL Remote for
Adaptive Server
Enterprise

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

- ◆ **Windows NT/2000/XP** All message protocols.
- ◆ **Sun Microsystems Solaris/Sparc** File sharing, FTP, and SMTP/POP only.

SQL Remote for
Adaptive Server
Anywhere

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

- ◆ **Windows 95/98/Me** All message links.
- ◆ **Windows NT/2000/XP** All message links.
- ◆ **Windows CE** FILE, FTP, and SMTP/POP links. For the file link, *dbremote* looks in *|My Documents|Synchronized Files*. On the desktop machine, the *SQLREMOTE* environment variable or directory message link parameter for the FILE link should be set to the following:

```
%SystemRoot%\Profiles\userid\Personal\ce-machine-name\
Synchronized Files
```

where *userid* and *ce-machine-name* are set to the appropriate values. With this setup, ActiveSync automatically synchronizes the message files between the desktop and CE system.

Check Mobile Devices ► Tools ► ActiveSync Options to ensure that file synchronization is activated.

☞ For information on setting message link parameters, see [“The file message system” on page 215](#).

- ◆ **Sun Microsystems Solaris/Sparc** File sharing, FTP, and SMTP/POP only.
- ◆ **Novell NetWare** File sharing, FTP, and SMTP/POP only.
- ◆ **Linux** File sharing, FTP, and SMTP/POP only.

For details of the supported UNIX operating system versions, see the *SQL Anywhere Studio Read Me First for UNIX*.

Index

Symbols

-b option	
Message Agent	224
-l option	
Message Agent	225
-m option	
Message Agent	229
-rd option	
Message Agent	230
-ro option	
Message Agent	226
-rp option	
Message Agent	230, 231
-rt option	
Message Agent	226
-u option	
Message Agent	225
#hook_dict table	
dbremote	318
unique primary keys	131

A

ActiveSync	
Windows CE	445
Adaptive Server Anywhere	
creating an Enterprise-compatible database	74
Adaptive Server Enterprise	
SQL Remote setup	21
tutorial with SQL Remote	53
adding	
articles	99
addresses	
file sharing	216
ftp	216
setting for publisher	210
SMTP	218
SMTP/POP	220
administering	
SQL Remote	13, 242
SQL Remote for Adaptive Server Anywhere	241, 242

SQL Remote for Adaptive Server Enterprise	263
SQL Remote overview	200
altering	
message types	211, 212
publications	99
article creation wizard	
using	99
article SQL Remote table	
Adaptive Server Anywhere	324
Adaptive Server Enterprise	336
articlecol SQL Remote table	
Adaptive Server Enterprise	337
definition	325
articlecols SQL Remote view	
Adaptive Server Anywhere	331
Adaptive Server Enterprise	344
articles	
adding	99
column-wise partitioning	143
creating	93
notes on	146
properties	93
row-wise partitioning	144
system table for	324, 336
valid	103, 147
whole table	143
articles SQL Remote view	
Adaptive Server Anywhere	331
Adaptive Server Enterprise	344

B

backups	
for remote databases	257
for replication	249, 253, 272
SQL Remote	225
batch mode	
Message Agent	223
batches	
passthrough mode	262
binary large objects	
replication	83, 314

- BLOB_THRESHOLD option
 - replication option 314
- BLOBs
 - replication 83, 314
- C**
- cache
 - for messages 229
- ccMail
 - SQL Remote 210
- character sets
 - compatible 74
 - conversions 75
 - SQL Remote 75
- collations
 - SQL Remote 75
- columns
 - publishing selected columns 94
- command line
 - environment variables 294
 - Message Agent 294
- COMMIT statement
 - event-hook procedures 318
 - replication 78
- compatibility
 - Adaptive Server Enterprise and Adaptive Server Anywhere 74, 196, 197
 - among databases 74
- COMPRESSION option
 - replication option 314
- configuration file
 - Message Agent options 294
- conflict detection
 - about 166
 - long data types 83
 - SQL Remote 79, 121
- conflict resolution
 - #remote 170
 - approaches to 124–126
 - example 168, 170
 - implementing 122, 166
 - limitations 167
 - triggers 122, 124, 166
- conflicts
 - #remote 170
 - approaches to resolving 124–126
 - avoiding 88
 - detection in SQL Remote 88
 - example 168, 170
 - locking 103, 147
 - managing 120, 165
 - not errors 245, 271
 - not in Message Agent output 245, 271
 - primary key 129, 134, 175
 - replication 88
 - reporting 126
 - resolving 122, 124, 166
 - SQL Remote handling of 121, 166
 - SQL Remote 88
 - VERIFY_ALL_COLUMNS option 123
- connections
 - Message Agent 224
- CONSOLIDATE permissions
 - granting 204, 207
 - managing 201
 - revoking 204
- consolidated databases
 - setting up (tutorial) 41, 59
 - tutorial for Adaptive Server Anywhere 34
- constraints
 - extraction utility 197
- continuous mode
 - Message Agent 223
- control statements
 - replication of 262
- conventions
 - documentation xii
- create a new remote user wizard
 - using 36, 205
- create database wizard
 - creating an Enterprise-compatible database 74
 - using 32
- CREATE statements
 - replication 82
- CREATE SUBSCRIPTION statement
 - about 139, 198
- creating
 - articles 93, 99
 - articles with column-wise partitioning 143

- articles with row-wise partitioning 144
- message types 211, 212
- publications 43, 93, 143
- publications (tutorial) 37
- publications with column-wise partitioning 94, 143
- publications with row-wise partitioning 95, 144
- publications with whole tables 93, 143
- subscriptions 44, 63, 139, 181
- subscriptions (tutorial) 38
- CURRENT PUBLISHER**
 - table for 336
 - tutorial 36, 42
- CURRENT REMOTE USER**
 - conflict resolution 170
 - special constant 124
 - table for 336
- cursors
 - passthrough mode 262
 - replication and 262
- D**
- daemon
 - dbremote 299
 - Message Agent 299
 - ssremote 299
- data recovery
 - SQL Remote 224
- data types
 - replication 83
- database extraction utility
 - SQL Remote 303
- databases
 - loading data into 191
 - procedures before extracting 192
 - synchronizing (tutorial) 45
- dates
 - replication 84, 315
- dbcc settrunc
 - using 273
- dbo user
 - system objects 302
- dbremote
 - #hook_dict table 318
 - about 223, 292
 - command 292
 - introduction 9
 - security 243
 - tutorial 47, 48, 67
- dbunload utility
 - replication 258
- dbxtract utility
 - about 189, 303
 - introduction 45
 - sp_hook_dbxtract_begin procedure 131
 - using 191
- DDL statements
 - replication of 82
 - SQL Remote 275
- debug control parameter
 - file message type 216
 - FTP message type 216
 - MAPI message type 221
 - SMTP message type 219
 - VIM message type 222
- defaults
 - extraction utility 197
- DELETE statement
 - replication of 78
- DELETE_OLD_LOGS option
 - managing transaction logs 253
 - replication option 314
- deleting
 - message types 212
- Deleting Corrupt Message error 235
- deploying
 - SQL Remote databases 185–187
- design
 - at the consolidated database 74
 - conflicts and publications 88
 - errors and publications 88
 - locking 103, 147
 - many-to-many example 112, 115, 157
 - many-to-many relationships 112, 157
 - performance for SQL Remote 103
 - publications 102, 127, 147, 173
 - SQL Remote 73
 - SQL Remote overview 92
 - SQL Remote for Adaptive Server Anywhere 91
 - SQL Remote for Adaptive Server Enterprise 141

SQL Remote for Adaptive Server		SQL statements and replication	89
Enterprise overview	142	types of in replication	88
SQL Remote overview	74	event hooks	
differences		commits not allowed	318
SQL Remote versions	437	rollbacks not allowed	318
directory control parameter		sp_hook_dbremote_begin stored	
file message type	216	procedure	318
documentation		sp_hook_dbremote_end stored	
conventions	xii	procedure	318
SQL Anywhere Studio	x	sp_hook_dbremote_message_apply_-	
DROP PUBLICATION statement		begin stored procedure	
using	100	320	
dropping		sp_hook_dbremote_message_apply_-	
message types	212, 213	end stored procedure	
publications	100	321	
dsi_num_threads		sp_hook_dbremote_message_missing	
Replication Server parameter	285	stored procedure	320
dsi_sql_data_style parameter		sp_hook_dbremote_message_sent	
configuring	286	stored procedure	320
dumps		sp_hook_dbremote_receive_begin	
coordinating	287	stored procedure	319
E		sp_hook_dbremote_receive_end	
e-mail		stored procedure	319
MAPI	210	sp_hook_dbremote_send_begin stored	
SMTP	210	procedure	320
VIM	210	sp_hook_dbremote_send_end stored	
encoding		procedure	320
about	235	sp_hook_dbremote_shutdown stored	
custom	236	procedure	319
encryption		sp_hook_ssrmmt_begin stored	
of messages	226	procedure	318
environment variable option		sp_hook_ssrmmt_end stored procedure	
Message Agent	294	318	
environment variables		sp_hook_ssrmmt_message_apply_begin	
SQLREMOTE	214	stored procedure	320
error handling		sp_hook_ssrmmt_message_apply_end	
about	246, 271	stored procedure	321
default	245, 271	sp_hook_ssrmmt_message_missing	
ignoring errors	245	stored procedure	320
errors		sp_hook_ssrmmt_message_sent stored	
conflicts are not	245, 271	procedure	320
handling	246, 271	sp_hook_ssrmmt_receive_begin stored	
ignoring	245	procedure	319
notification	246, 271	sp_hook_ssrmmt_receive_end stored	
primary key replication	88	procedure	319
reporting	245, 271	sp_hook_ssrmmt_send_begin stored	
		procedure	320

- sp_hook_ssrmmt_send_end stored procedure 320
 - sp_hook_ssrmmt_shutdown stored procedure 319
 - synchronization 318
 - extract database wizard
 - extracting a remote database in Sybase Central 302
 - using 39, 193
 - extracting
 - custom procedures 193
 - databases 185, 186, 189
 - designing a procedure 193
 - many databases 193
 - operating systems 189
 - performance 155, 162
 - procedures before 192
 - reload files 191
 - using Sybase Central 192
 - extraction utility 191
 - about 189, 196, 197
 - for Adaptive Server Enterprise 196, 197
 - groups 195
 - limits 195
 - options 317
 - procedures before using 192
 - purpose 195
 - using from Sybase Central 192
- F**
- feedback
 - documentation xvi
 - providing xvi
 - file sharing
 - control parameters 216
 - message type 210, 215
 - FIRE_TRIGGERS option
 - trigger actions 81
 - Force_Download control parameter
 - MAPI message type 221
 - foreign key creation wizard
 - using 33
 - foreign keys
 - publications 102, 105, 147, 149, 152
 - territory realignment 107, 108, 151
 - frequency
 - of sending 206
 - ftp
 - control parameters 216
 - message type 210, 216
 - troubleshooting 217
- G**
- generating
 - unique column values 129
 - global autoincrement
 - using to generate unique values 129
 - GLOBAL_DATABASE_ID option
 - setting 130
 - GRANT CONSOLIDATE statement 204
 - GRANT PUBLISH statement 201, 203
 - consolidated database (tutorial) 42
 - tutorial 35
 - GRANT REMOTE statement 204
 - consolidated database (tutorial) 42
 - tutorial 35
 - granting
 - CONSOLIDATE permissions 204
 - publish permissions 201, 203
 - remote permissions 204
 - groups
 - extracting 195, 305
- H**
- host control parameter
 - FTP message type 216
- I**
- icons
 - used in manuals xiv
 - IF statement
 - passthrough mode 262
 - replication of 262
 - IMAGE data type
 - replication 83
 - INSERT statement
 - replication of 78
 - Internet
 - e-mail 218
 - SQL Remote 218
 - IPM_Receive control parameter
 - MAPI message type 221

- IPM_Send control parameter
 - MAPI message type 221
- L**
- laptop computers
 - replication 15
 - SQL Remote 15
- limitations
 - conflict resolution 167
 - Enterprise to Enterprise 440
- loading databases 191
- locking
 - in a replication system 103, 147
 - publication design 129, 134, 175
- log management
 - SQL Remote 225
- log transfer interface
 - the Message Agent 273
- LONG BINARY data type
 - replication 83
- LONG VARCHAR data type
 - replication 83
- LOOP statement
 - passthrough mode 262
- Lotus Notes
 - SQL Remote 210, 221, 222
- LTM
 - SQL Remote 265
- M**
- maintenance releases
 - upgrading 187
- many-to-many relationships
 - example 112, 114, 157, 158
 - publication design 112, 157
 - SUBSCRIBE_BY_REMOTE option 118
 - Subscribe_by_remote option 164
 - territory realignment 115
 - triggers for 116
- MAPI
 - control parameters 220, 221
 - message type 210
- marker SQL Remote table
 - Adaptive Server Enterprise 337
- media failure
 - SQL Remote 224
- Message Agent
 - l option 225
 - m option 229
 - rd option 230
 - ro option 226
 - rp option 230, 231
 - rt option 226
 - u option 225
 - about 223, 269
 - batch mode 223
 - command 292
 - connections 224
 - continuous mode 223
 - delivering messages 237, 239
 - introduction 9
 - message tracking 237, 239
 - output 245, 271
 - performance 228, 232
 - polling 230
 - resend requests 230
 - running 242
 - running as a service 242
 - schema changes 275
 - security 226, 243, 269
 - settings 225
 - SQL Remote administration 200
 - starting 242
 - subscription processing 87
 - threading 228
 - transaction log management 249, 253, 257, 272
 - trigger replication 81
 - tutorial 47, 48, 66, 67
 - user IDs 269
 - worker threads 228
- message link parameters 214
 - EXTERNAL_REMOTE_OPTIONS
 - replication option 314
- message links
 - supported 444
- message systems
 - supported 444
- message tracking
 - SQL Remote administration 200
- message type creation wizard
 - using 211
- message types

-
- about 210
 - altering 211, 212
 - creating 211, 212
 - dropping 212, 213
 - editing properties 211
 - file sharing 215, 216
 - ftp 216
 - MAPI 220, 221
 - parameters 214
 - SMTP 219
 - SQL Remote administration 200
 - VIM 221, 222
 - working with 210
 - messages
 - caching 229
 - compression 235, 314
 - controlling size 83
 - custom encoding 236
 - delivering 237, 239
 - encoding 235
 - in SQL Remote 237, 239
 - receiving (tutorial) 48, 67
 - resending 230
 - sending (tutorial) 47, 66
 - synchronizing databases 198
 - tracking 237, 239
 - Microsoft Exchange
 - profile 221
 - missing messages
 - about 231
 - mobile workforces
 - publication design 97, 145
 - SQL Remote 15
 - SQL Remote for 13
 - multi-tier installations
 - passthrough statements 262
 - permissions 208
 - N**
 - named constraints
 - extraction utility 197
 - named defaults
 - extraction utility 197
 - NCHAR data type
 - extraction utility 197
 - NetWare
 - SQL Remote 216
 - supported message types 210
 - newsgroups
 - technical support xvi
 - Notes
 - SQL Remote 210, 221
 - Novell NetWare
 - availability on 445
 - NVARCHAR data type
 - extraction utility 197
 - O**
 - object SQL Remote table
 - Adaptive Server Enterprise 338
 - offsets
 - in transaction log 237
 - operating systems
 - availability on 445
 - supported 443
 - option SQL Remote table
 - Adaptive Server Enterprise 338
 - options
 - BLOB_THRESHOLD 313
 - COMPRESSION 313
 - DELETE_OLD_LOGS 313
 - EXTERNAL_REMOTE_OPTIONS 313
 - list of SQL Remote 313
 - QUALIFY_OWNERS 313
 - QUOTE_ALL_IDENTIFIERS 313
 - REPLICATION_ERROR 313
 - SAVE_REMOTE_PASSWORDS 313
 - SR_DATE_FORMAT 313
 - SR_TIME_FORMAT 313
 - SR_TIMESTAMP_FORMAT 313
 - SUBSCRIBE_BY_REMOTE 118, 164, 313
 - the extraction utility 317
 - VERIFY_ALL_COLUMNS 313
 - VERIFY_THRESHOLD 83, 313
 - OUTPUT_LOG_SEND_LIMIT remote
 - option 226
 - OUTPUT_LOG_SEND_NOW remote
 - option 226
 - OUTPUT_LOG_SEND_ON_ERROR
 - remote option

- using 226
- P**
- partitioning
 - column-wise 94, 143
 - row-wise 95, 144
- passthrough mode
 - Adaptive Server Enterprise 276
 - batches 262
 - for Adaptive Server Anywhere 260
 - operations not replicated 262
 - operations not replicated in 262
 - uses 261
- passthrough SQL Remote table
 - Adaptive Server Enterprise 338
- PASSTHROUGH statement
 - using 260
- passthrough statements
 - multi-tier installations 262
- password control parameter
 - FTP message type 216
 - VIM message type 222
- passwords
 - extraction utility 197
 - saving 315
- Path control parameter
 - VIM message type 222
- patience
 - Message Agent 231
- performance
 - database extraction 193
 - design tips for SQL Remote 103
 - incoming messages 230
 - Message Agent 228
 - message sending 232
 - number of subscriptions 87
 - publications 86
 - replication throughput 228
 - replication turnaround time 228
 - SQL Remote 228
 - ssxtract 155, 162
 - threading 228
- permissions
 - CONSOLIDATE 204
 - granting CONSOLIDATE 207
 - multi-tier installations 208
 - publish 201, 203
 - publish (tutorial) 35, 42
 - remote 204
 - remote (tutorial) 35, 42
 - revoking REMOTE 207
 - SQL Remote administration 200
- platforms
 - supported operating systems 443
- polling
 - messages 230
- pop3_host control parameter
 - SMTP message type 219
- pop3_password control parameter
 - SMTP message type 219
- pop3_userid control parameter
 - SMTP message type 219
- port control parameter
 - FTP message type 216
- primary key pools
 - about 175
 - generating unique values using default global autoincrement 129
 - replenishing 135, 177
 - replicating 134, 176
 - summary 138, 180
- primary keys
 - generating unique values 129
 - generating unique values using pools 133
- publications 78, 102, 147
- replication 127, 129, 173
- replication errors 88
- SQL Remote 127, 173
- uniqueness 175
- procedure groups
 - extraction utility 197
- procedures
 - passthrough mode 262
 - replicating 80
 - SQL Remote 80
 - SQL Remote Open Server 287
 - statements corresponding to 438
- profiles
 - Microsoft Exchange 221
- properties
 - articles 93
 - message type properties 211
 - publications 93

- publication creation wizard
 - creating an article using a subscription expression 98
 - creating SQL Remote publications 37, 93
 - creating SQL Remote publications using a WHERE clause 96
 - creating SQL Remote publications with column-wise partitioning 94
- publication design
 - Adaptive Server Anywhere 91
 - for many subscribers 97, 145
 - SQL Remote for Adaptive Server Enterprise 141
 - using subscription expression 97, 145
- publication SQL Remote table
 - Adaptive Server Anywhere 326
 - Adaptive Server Enterprise 339
- publications
 - about 11
 - altering 99
 - column-wise partitioning 94, 143
 - creating 43, 93, 143
 - creating (tutorial) 37
 - design at consolidated database 92
 - designing 73, 74, 88, 102, 127, 147, 173
 - dropping 100
 - example 51
 - foreign keys 102, 105, 147, 149
 - locking 103, 147
 - managing subscriptions 139
 - many-to-many example 112, 114, 115, 157, 158
 - many-to-many relationships 112, 157
 - notes 101
 - performance 86, 103
 - primary keys 102, 127, 129, 134, 147, 173, 175
 - properties 93
 - referential integrity 127, 173
 - row-wise partitioning 95, 144
 - setting up 93, 143
 - simple 93, 143
 - subqueries 105
 - tables in many publications 86
 - transactions 103, 147
 - using a WHERE clause 96, 145
 - whole table 93
- publications SQL Remote view
 - Adaptive Server Anywhere 331
 - Adaptive Server Enterprise 344
- publish permissions
 - granting 201, 203
 - granting (tutorial) 35
 - managing 201
 - remote permissions (tutorial) 35, 42
 - revoking 201, 203
- publish permissions (tutorial) 42
- publisher
 - about 201, 203
 - adding to a database (tutorial) 42
 - address 210
 - creating 201, 203
 - tutorial 36
- publisher SQL Remote table
 - Adaptive Server Enterprise 339
- publishing
 - selected columns 94
- Q**
- QUALIFY_OWNERS option
 - replication option 314
- quiet system
 - definition 275
- QUOTE_ALL_IDENTIFIERS option
 - replication option 315
- R**
- receive_all control parameter
 - VIM message type 222
- receiving messages
 - (tutorial) 48
 - tutorial 67
- recovery
 - SQL Remote 224
- referential integrity
 - replication 127, 173
 - SQL Remote 127, 173
- registry
 - SQL Remote 214
- reload files
 - database extraction 191

- remote databases
 - remote permissions 204
 - setting up (tutorial) 38, 44, 45
- remote permissions
 - granting 204
 - managing 201
 - revoking 204, 207
 - Sybase Central 204
- remotoption SQL Remote table
 - Adaptive Server Anywhere 326
- remoteoptions SQL Remote view
 - Adaptive Server Anywhere 332
 - Adaptive Server Enterprise 345
- remoteoptiontype SQL Remote table
 - Adaptive Server Anywhere 326
- remotetable table
 - about 340
- remotetables SQL Remote view
 - Adaptive Server Enterprise 345
- remotetype SQL Remote table
 - about 340
 - Adaptive Server Anywhere 327
- remotetypes SQL Remote view
 - Adaptive Server Enterprise 345
- remoteuser SQL Remote table
 - about 341
 - Adaptive Server Anywhere 327
- remoteuser table 237
- remoteusers SQL Remote view
 - Adaptive Server Anywhere 332
 - Adaptive Server Enterprise 346
- replication
 - administering 13
 - backup procedures 249, 253, 257, 272
 - blobs 83
 - case studies 15, 16
 - conflicts 88
 - data definition statements 82
 - data types 83
 - dbremote 292
 - design 102, 147
 - design overview 102
 - Message Agent 292
 - mobile workforces 15
 - of control statements 262
 - of cursor operations 262
 - of cursor statements 262
 - of SQL statements 260
 - of stored procedures 262
 - options 313
 - passthrough mode 260
 - primary key errors 127, 173
 - primary keys 129, 134, 175
 - procedures 80
 - publications 11
 - referential integrity errors 127, 173
 - server-to-laptop replication 15
 - server-to-server 16
 - setup examples 15
 - ssqueue 310
 - ssremote 292
 - subscriptions 11
 - synchronization 303
 - transaction log 78
 - transaction log and 13
 - transaction log management 249, 253, 257, 272
 - triggers 80, 127
 - upgrading databases 258
- Replication Agent
 - SQL Remote 265, 278, 279
- replication conflicts
 - about 165
 - managing 120
- replication definitions
 - Replication Server 285
- replication options
 - QUALIFY_OWNERS 314
 - QUOTE_ALL_IDENTIFIERS 315
 - REPLICATION_ERROR 89, 246, 271, 315
 - SAVE_REMOTE_PASSWORDS 315
 - SUBSCRIBE_BY_REMOTE 316
 - VERIFY_ALL_COLUMNS 317
 - VERIFY_THRESHOLD 317
- replication role
 - granting 269
- Replication Server
 - configuring 285
 - restart the connection 286
 - SQL Remote 265, 277, 278, 285
 - SQL Remote architecture 279
 - ssqueue 282
 - using SQL Remote with 310

-
- REPLICATION_ERROR option
 - and error handling procedures 246
 - error handling 271
 - replication option 315
 - tracking SQL errors 89
 - reporting
 - conflicts 126
 - errors 271
 - reporting errors 245, 271
 - resend requests
 - about 231
 - messages 230
 - resetting
 - subscriptions 432
 - RESOLVE UPDATE triggers 122, 124
 - REVOKE CONSOLIDATE statement 204
 - REVOKE PUBLISH statement 201, 203
 - REVOKE REMOTE statement 204, 207
 - revoking
 - CONSOLIDATE permissions 204
 - publish permissions 201, 203
 - remote permissions 204
 - revoking remote permissions 207
 - roles
 - extraction utility 197
 - ROLLBACK statement
 - event-hook procedures 318
 - root control parameter
 - FTP message type 216
 - rs_dumpdb
 - using 287
 - rs_dumptran
 - using 287
 - running
 - the Message Agent 242
 - S**
 - salespub.sql
 - example publication 51
 - SAVE_REMOTE_PASSWORDS option
 - replication option 315
 - schema changes
 - SQL Remote 275
 - SQL Remote Open Server 287
 - SEND AT
 - frequency setting 206
 - SEND EVERY
 - frequency setting 206
 - send frequency
 - selecting 206
 - the Message Agent 223
 - send_vim_mail control parameter
 - VIM message type 222
 - sending messages
 - tutorial 47, 66
 - server-to-server replication 16
 - services
 - Message Agent as 242
 - setting up
 - consolidated database (tutorial) 34
 - consolidated databases (tutorial) 41, 59
 - publications 93
 - remote databases (tutorial) 38, 44
 - subscriptions 139
 - setup
 - SQL Remote for Adaptive Server
 - Enterprise 19, 21
 - SQL Remote for Adaptive Server
 - Enterprise overview 20
 - SQL Remote for Adaptive Server
 - Enterprise stable queue 23
 - TEMPDB for SQL Remote 21
 - SMTP
 - control parameters 219
 - e-mail 218
 - message type 210, 218
 - SQL Remote 218
 - SMTP/POP
 - addresses 220
 - smtp_authenticate control parameter
 - SMTP message type 219
 - smtp_host control parameter
 - SMTP message type 219
 - smtp_password control parameter
 - SMTP message type 219
 - smtp_userid control parameter
 - SMTP message type 219
 - software
 - dbremote 292
 - dbxtract 303
 - ssqueue 310
 - ssremote 292
 - ssxtract 303
 - sp_add_article procedure

syntax	379	procedure	
sp_add_article_col procedure		SQL syntax	320
syntax	381	sp_hook_dbremote_shutdown stored	
sp_add_remote_table procedure		procedure	
syntax	382	SQL syntax	319
sp_create_publication procedure		sp_hook_dbxtract_begin procedure	
syntax	384	unique primary keys	131
sp_drop_publication procedure		using	131
syntax	385	sp_hook_ssrmmt_begin stored procedure	
sp_drop_remote_type procedure		SQL syntax	318
syntax	386	sp_hook_ssrmmt_end stored procedure	
sp_drop_sql_remote procedure		SQL syntax	318
syntax	387	sp_hook_ssrmmt_message_apply_begin	
uninstalling SQL Remote for Adaptive		stored procedure	
Server Enterprise	26	SQL syntax	320
sp_grant_consolidate procedure		sp_hook_ssrmmt_message_apply_end	
syntax	388	stored procedure	
sp_grant_remote procedure		SQL syntax	321
syntax	391	sp_hook_ssrmmt_message_missing stored	
sp_hook_dbremote_begin stored		procedure	
procedure		SQL syntax	320
SQL syntax	318	sp_hook_ssrmmt_message_sent stored	
sp_hook_dbremote_end stored procedure		procedure	
SQL syntax	318	SQL syntax	320
sp_hook_dbremote_message_apply_-		sp_hook_ssrmmt_receive_begin stored	
begin stored		procedure	
procedure		SQL syntax	319
SQL syntax	320	sp_hook_ssrmmt_receive_end stored	
sp_hook_dbremote_message_apply_end		procedure	
stored procedure		SQL syntax	319
SQL syntax	321	sp_hook_ssrmmt_send_begin stored	
sp_hook_dbremote_message_missing		procedure	
stored procedure		SQL syntax	320
SQL syntax	320	sp_hook_ssrmmt_send_end stored	
sp_hook_dbremote_message_sent stored		procedure	
procedure		SQL syntax	320
SQL syntax	320	sp_hook_ssrmmt_shutdown stored	
sp_hook_dbremote_receive_begin stored		procedure	
procedure		SQL syntax	319
SQL syntax	319	sp_link_option procedure	
sp_hook_dbremote_receive_end stored		syntax	394
procedure		sp_modify_article procedure	
SQL syntax	319	syntax	396
sp_hook_dbremote_send_begin stored		sp_modify_remote_table procedure	
procedure		syntax	398
SQL syntax	320	sp_passthrough procedure	
sp_hook_dbremote_send_end stored		about	276

syntax	400	syntax	420
sp_passthrough_piece procedure		sp_queue_transaction procedure	
about	276	syntax	421
syntax	401	sp_remote procedure	
sp_passthrough_stop procedure		syntax	422
about	276	sp_remote_option procedure	
syntax	403	syntax	423
sp_passthrough_subscription procedure		sp_remote_type procedure	
about	276	syntax	425
syntax	404	sp_remove_article procedure	
sp_passthrough_user procedure		syntax	426
about	276	sp_remove_article_col procedure	
syntax	405	syntax	427
sp_populate_sql_anywhere procedure		sp_remove_remote_table procedure	
about	197	syntax	428
syntax	406	sp_revoke_consolidate procedure	
sp_publisher procedure		syntax	429
syntax	407	sp_revoke_remote procedure	
sp_queue_clean procedure		syntax	430
syntax	408	sp_setreplicate procedure	
sp_queue_confirmed_delete_old		sp_add_remote_table	382
procedure		sp_subscription procedure	
syntax	409	about	181
sp_queue_confirmed_transaction		syntax	431
procedure		using	198
syntax	410	sp_subscription_reset procedure	
sp_queue_delete_old procedure		syntax	432
syntax	411	sp_user_extraction_hook	
sp_queue_drop procedure		example	161
syntax	412	sp_user_extraction_hook procedure	
uninstalling SQL Remote stable queue		about	197
26		SQL Anywhere Studio	
sp_queue_dump_database procedure		documentation	x
syntax	413	SQL Remote	
sp_queue_dump_transaction procedure		administering	13, 261
syntax	414	articles	324
sp_queue_get_state procedure		backup procedures	249, 253, 257, 272
syntax	415	case studies	15, 16
sp_queue_log_transfer_reset procedure		components	8
syntax	416	concepts	7
sp_queue_read procedure		creating publications	43
syntax	417	creating publications (tutorial)	37
sp_queue_reset procedure		dbremote (tutorial)	47, 48, 67
syntax	418	dbxtract utility	303
sp_queue_set_confirm procedure		design overview	92
syntax	419	granting publish permissions (tutorial)	
sp_queue_set_progress procedure		35, 42	

granting remote permissions (tutorial)		SQL Remote procedures	
35, 42		sp_add_article	379
Message Agent (tutorial)	47, 48, 66, 67	sp_add_article_col	381
Message Agent introduction	9	sp_add_remote_table	382
message delivery	237, 239	sp_create_publication	384
message tracking	237, 239	sp_drop_publication	385
message types for Windows CE	212	sp_drop_remote_type	386
mobile workforces	13, 15	sp_drop_sql_remote	387
multi-tier installations	262	sp_grant Consolidate	388
options	313	sp_grant_remote	391
publications	11	sp_link_option	394
server-to-laptop replication	15	sp_modify_article	396
server-to-server replication	16	sp_modify_remote_table	398
setting up (tutorial)	57	sp_passthrough	400
setting up a consolidated database	34, 41, 59	sp_passthrough_piece	401
setting up a remote database	44	sp_passthrough_stop	403
setting up a remote database (tutorial)	38	sp_passthrough_subscription	404
setup examples	15	sp_passthrough_user	405
setup for Adaptive Server Enterprise	20, 21	sp_populate_sql_anywhere	406
ssremote (tutorial)	66	sp_publisher	407
ssxtract utility	303	sp_queue_clean	408
subscribers	13	sp_queue_confirmed_delete_old	409
subscriptions	11	sp_queue_confirmed_transaction	410
system tables	324	sp_queue_delete_old	411
TEMPDB	21	sp_queue_drop	412
transaction log management	249, 253, 257, 272	sp_queue_dump_database	413
tutorial for Adaptive Server Enterprise	53	sp_queue_dump_transaction	414
uninstalling	387, 412	sp_queue_get_state	415
unloading databases	258	sp_queue_log_transfer_reset	416
upgrading databases	258	sp_queue_read	417
upgrading for Adaptive Server Enterprise	25	sp_queue_reset	418
user IDs	269	sp_queue_set_confirm	419
SQL remote administration	199	sp_queue_set_progress	420
SQL Remote Open Server		sp_queue_transaction	421
architecture	279	sp_remote	422
command line	310	sp_remote_option	423
IRIX	445	sp_remote_type	425
procedures	287	sp_remove_article	426
schema changes	275	sp_remove_article_col	427
setting up	282	sp_remove_remote_table	428
when needed	278	sp_revoke Consolidate	429
		sp_revoke_remote	430
		sp_subscription	431
		sp_subscription_reset	432
		SQL Remote system tables	
		#remote	336
		article	324, 336

-
- articlecol 325, 337
 - marker 337
 - object 338
 - option 338
 - passthrough 338
 - publication 326, 339
 - publisher 339
 - remoteoption 326
 - remoteoptiontype 326
 - remotetable 340
 - remotetype 327, 340
 - remoteuser 327, 341
 - sr_article 336
 - sr_remoteoption 339
 - sr_remoteoptiontype 340
 - subscription 329, 343
 - SQL Remote system views
 - articlecols 331, 344
 - articles 331, 344
 - publications 331, 344
 - remoteoptions 332, 345
 - remotetables 345
 - remotetypes 345
 - remoteusers 332, 346
 - subscriptions 333, 347
 - SQL statements
 - replication 82
 - replication of 78
 - SQL syntax
 - sp_hook_dbremote_begin stored procedure 318
 - sp_hook_dbremote_end stored procedure 318
 - sp_hook_dbremote_message_apply_-begin stored procedure 320
 - sp_hook_dbremote_message_apply_-end stored procedure 321
 - sp_hook_dbremote_message_missing stored procedure 320
 - sp_hook_dbremote_message_sent stored procedure 320
 - sp_hook_dbremote_receive_begin stored procedure 319
 - sp_hook_dbremote_receive_end stored procedure 319
 - sp_hook_dbremote_send_begin stored procedure 320
 - sp_hook_dbremote_send_end stored procedure 320
 - sp_hook_dbremote_shutdown stored procedure 319
 - sp_hook_ssrmmt_begin stored procedure 318
 - sp_hook_ssrmmt_end stored procedure 318
 - sp_hook_ssrmmt_message_apply_begin stored procedure 320
 - sp_hook_ssrmmt_message_apply_end stored procedure 321
 - sp_hook_ssrmmt_message_missing stored procedure 320
 - sp_hook_ssrmmt_message_sent stored procedure 320
 - sp_hook_ssrmmt_receive_begin stored procedure 319
 - sp_hook_ssrmmt_receive_end stored procedure 319
 - sp_hook_ssrmmt_send_begin stored procedure 320
 - sp_hook_ssrmmt_send_end stored procedure 320
 - sp_hook_ssrmmt_shutdown stored procedure 319
 - SQLANY.INI
 - SQL Remote 214
 - SQLREMOTE environment variable
 - alternative to 216
 - setting message control parameters 214
 - SQL Remote
 - about 4
 - about the manual 5
 - squpdate.sql
 - upgrading the stable queue 25
 - sr_article table
 - about 336
 - sr_articlecol table
 - about 337
 - sr_confirmed_transaction table
 - about 350
 - sr_marker table
 - about 337
 - sr_object table

about	338	about	189, 303
sr_option table		using	191, 196, 197
about	338	stable queue	
sr_passthrough table		cleaning	296
about	338	Replication Server	279
sr_publication table		setup	23
about	339	SQL Remote Open Server	279
sr_publisher SQL Remote table		ssqueue	279
about	339	system tables	348
sr_queue_coordinate table		stableq.sql	
about	350	SQL Remote setup	23
sr_queue_state table		starting	
about	348	the Message Agent	242
sr_remotoption SQL Remote table		statements	
about	339	replication of	78
sr_remotoptiontype SQL Remote table		stored procedures	
Adaptive Server Enterprise	340	passthrough mode	262
sr_remotetable table		replication of	80
about	340	SQL Remote Open Server	287
sr_remotetype table		subqueries	
about	340	publications	105
sr_remotouser table		SQL Remote	105
about	341	SUBSCRIBE_BY_REMOTE option	
sr_subscription table		many-to-many relationships	118, 164
about	343	replication option	316
sr_transaction table		subscription expressions	
about	349	about	97, 145
ssqueue		cost of evaluating	86
about	277, 310	in the transaction log	86
architecture	279	many-valued	86
command line	310	subqueries	105
IRIX	445	subscription SQL Remote table	
setting up	282	about	343
when needed	278	Adaptive Server Anywhere definition	
ssremote		329	
about	223, 269, 292	subscription views	
command	292	about	162
introduction	9	subscription-list columns	
Message Agent (tutorial)	66	about	152
security	269	in SQL Remote	149
tutorial	66	maintaining	154, 159, 161
ssremote.sql		triggers	159
SQL Remote setup	21	triggers for	154
ssupdate.sql		subscriptions	
upgrading SQL Remote for Adaptive		about	11
Server Enterprise	25	creating	44, 63, 139, 181
ssxtract utility		creating (tutorial)	38

-
- | | | | |
|------------------------------------|----------|------------------------------|---------------|
| managing | 139 | technical support | |
| Message Agent | 87 | newsgroups | xvi |
| performance | 87 | TEMPDB | |
| resetting | 432 | SQL Remote requirements | 21 |
| setting up | 139, 181 | territory realignment | |
| synchronizing | 191, 198 | about | 154 |
| subscriptions SQL Remote view | | foreign keys | 108, 151 |
| Adaptive Server Anywhere | 333 | many-to-many relationships | 115 |
| Adaptive Server Enterprise | 347 | replication of UPDATES | 79 |
| Sun Solaris | | subscription-list columns | 152 |
| availability on | 445 | triggers for | 116 |
| support | | territory realignments | |
| newsgroups | xvi | foreign keys | 107 |
| Sybase Central | | testing | |
| extraction utility | 192 | SQL Remote | 187 |
| message types | 212 | TEXT data type | |
| synchronization | | replication | 83 |
| about | 189 | threading | |
| customizing | 318 | Message Agent | 228 |
| databases | 185, 186 | times | |
| event hooks | 318 | replication | 84, 316 |
| operating systems | 189 | timestamp columns | |
| using dbxtract | 191 | extraction utility | 197 |
| using ssxtract | 191 | transaction log | |
| using the extraction utility | 191 | Adaptive Server Enterprise | 273 |
| SYNCHRONIZE SUBSCRIPTION | | managing | 273 |
| statement | | Message Agent | 292 |
| about | 198 | message tracking | 237 |
| synchronizing | | offsets | 237 |
| SQL Remote databases using the | | publications | 86 |
| message system | 198 | replication | 13, 78 |
| SYSREMOTEUSER table | 237 | scanning for SQL Remote Open | |
| system objects | | Server | 278 |
| the dbo user | 302 | SQL Remote | 13, 78, 292 |
| system objects for Adaptive Server | | update statements | 86 |
| Anywhere | 323 | transaction log mirror | |
| system objects for Adaptive Server | | replication | 249, 272 |
| Enterprise | 335 | transactions | |
| system tables | | replication | 78 |
| stable queue | 348 | triggers | |
| SYSARTICLE | 324, 336 | conflict resolution | 122, 124, 166 |
| | | replication | 80, 109, 127 |
| | | replication option | 81 |
| | | RESOLVE UPDATE | 122, 124 |
| | | SQL Remote | 80, 127 |
| | | subscription-list columns | 154, 159 |
| | | territory realignment | 109 |
-
- T**
- | | |
|--------------------------|---------|
| tables | |
| column-wise partitioning | 94, 143 |
| publishing | 93, 143 |
| row-wise partitioning | 95, 144 |

- troubleshooting
 - SQL Remote errors 226
- truncation point
 - in transaction log 273
 - setting 273
- turnaround time
 - replication performance 228
- tutorials
 - SQL Remote with Adaptive Server Enterprise 53
 - SQL Remote for Adaptive Server Anywhere 27
- U**
- uninstalling
 - from a database 387, 412
 - SQL Remote for Adaptive Server Enterprise 26
 - SQL Remote stable queue 26
- unique column values
 - generating 129
- unique values
 - generating using default global autoincrement 129
 - generating using pools 133
- UNIX
 - supported message types 210
- unlink_delay control parameter
 - file message type 216
- unloading
 - consolidated databases 258
- UPDATE conflicts
 - about 165
 - managing 120
- UPDATE statement
 - conflict detection 79, 83
 - publications 109
 - replication of 78, 107, 108, 151
 - territory realignment 79
- updates
 - information in transaction log 86
- upgrades
 - COMPRESSION option 235
 - SQL Remote 235
- upgrading
 - replication 258
 - SQL Remote 258
- SQL Remote installations 187
- SQL Remote for Adaptive Server Enterprise 25
- user control parameter
 - FTP message type 216
- user creation wizard
 - using 36, 201
- user IDs
 - extracting groups 195
 - Message Agent 269
- Userid control parameter
 - VIM message type 222
- utilities
 - extraction 192
- V**
- VERIFY_ALL_COLUMNS option
 - replication option 317
 - using 123
- VERIFY_THRESHOLD
 - replication option 317
- VERIFY_THRESHOLD option
 - message size 83
- VIM
 - control parameters 222
 - message type 210, 221
- W**
- WHERE clause
 - in publications 96, 145
- whole tables
 - publishing 93
- Windows
 - SQL Remote availability on supported SQL Remote message types 210
- Windows CE
 - ActiveSync 445
 - replication 445
 - SQL Remote message types 212
- wizards
 - article creation 99
 - create a new remote user 205
 - create database 32
 - extract database 39, 193, 302
 - foreign key creation 33
 - message type creation 211

publication creation 37, 93, 94, 96, 98
remote user creation 36
user creation 36, 201
worker threads
 Message Agent 228