SYBASE®

Sybase® Adaptive Server™ Enterprise Security Features User's Guide

Adaptive Server™

Document ID: 36052-01-1150

September 1997

Copyright Information

Copyright © 1989–1997 by Sybase, Inc. All rights reserved.

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement. 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 Sybase, Inc.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, the Sybase logo, APT-FORMS, Certified SYBASE Professional, Data Workbench, First Impression, InfoMaker, PowerBuilder, Powersoft, Replication Server, S-Designor, SQL Advantage, SQL Debug, SQL SMART, SQL Solutions, Transact-SQL, VisualWriter, and VQL are registered trademarks of Sybase, Inc. Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Monitor, ADA Workbench, AnswerBase, Application Manager, AppModeler, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, APT Workbench, Backup Server, BayCam, Bit-Wise, ClearConnect, Client-Library, Client Services, CodeBank, Column Design, Connection Manager, DataArchitect, Database Analyzer, DataExpress, Data Pipeline, DataWindow, DB-Library, dbQ, Developers Workbench, DirectConnect, Distribution Agent, Distribution Director, Dynamo, Embedded SQL, EMS, Enterprise Client / Server, Enterprise Connect, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Formula One, Gateway Manager, GeoPoint, ImpactNow, InformationConnect, InstaHelp, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Logical Memory Manager, MainframeConnect, Maintenance Express, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, MethodSet, Net-Gateway, NetImpact, Net-Library, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Client, Open ClientConnect, Open Client / Server, Open Client / Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, PB-Gen, PC APT-Execute, PC DB-Net, PC Net Library, Power++, Power AMC, PowerBuilt, PowerBuilt with PowerBuilder, PowerDesigner, Power J, PowerScript, PowerSite, PowerSocket, Powersoft Portfolio, Power Through Knowledge, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Quickstart Datamart, Replication Agent, Replication Driver, Replication Server Manager, Report-Execute, Report Workbench, Resource Manager, RW-DisplayLib, RW-Library, SAFE, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Anywhere, SQL Central, SQL Code Checker, SQL Edit, SQL Edit / TPU, SQL Modeler, SQL Remote, SQL Server, SQL Server / CFT, SQL Server / DBM, SQL Server Manager, SQL Server SNMP SubAgent, SQL Station, SQL Toolset, Sybase Client / Server Interfaces, Sybase Development Framework, Sybase Gateways, Sybase IQ, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase Virtual Server Architecture, Sybase User Workbench, SybaseWare, SyBooks, System 10, System 11, the System XI logo, SystemTools, Tabular Data Stream, The Architecture for Change, The Enterprise Client / Server Company, The Model for Client / Server Solutions, The Online Information Center, Translation Toolkit, Turning Imagination Into Reality, Unibom, Unilib, Uninull, Unisep, Unistring, Viewer, Visual Components, VisualSpeller, WarehouseArchitect, WarehouseNow, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, and XA-Server are trademarks of Sybase, Inc. 6 / 97

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Table of Contents

About This Book

1. Introduction

2. Logging into Adaptive Server

3. Roles in Adaptive Server

4. Granting Database Permissions

Index

List of Tables

About This Book

This manual, the *Security Features User's Guide*, describes the security features available in Sybase® Adaptive Server™ Enterprise release 11.5, and provides instructions and guidelines for using these features.

Audience

This manual is intended for all users of Adaptive Server, but it particularly addresses the non-administrative user who owns database objects but who is not a System Administrator or a System Security Officer. This guide assumes that readers have some general understanding of Adaptive Server and understand simple SQL statements. For information on setting up a secure operating environment for Adaptive Server, see the *Security Administration Guide*.

How to Use This Book

This manual contains the following chapters:

- Chapter 1, "Introduction," provides an overview of the security features that are available with Adaptive Server and directs you to the chapter where each feature is discussed in detail.
- Chapter 2, "Logging into Adaptive Server," discusses Adaptive Server login accounts, password management, and how to log in.
- Chapter 3, "Roles in Adaptive Server," describes the administrative and ownership roles that exist in Adaptive Server, as well as the types of roles that can be defined and created by a System Security Officer.
- Chapter 4, "Granting Database Permissions," describes the discretionary access control functions provided by Adaptive Server. The chapter describes how to use the **grant** and **revoke** commands to specify whether users or groups can access particular database objects and to specify which commands or database operations are available to an individual user.

Adaptive Server Enterprise Documents

The following documents comprise the Sybase Adaptive Server Enterprise documentation:

- The *Release Bulletin* for your platform – contains last-minute information that was too late to be included in the books.

  A more recent version of the *Release Bulletin* may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use SyBooks™-on-the-Web.

- The Adaptive Server installation documentation for your platform – describes installation and upgrade procedures for all Adaptive Server and related Sybase products.
- The Adaptive Server configuration documentation for your platform – describes configuring a server, creating network connections, configuring for optional functionality, such as auditing, installing most optional system databases, and performing operating system administration tasks.
- *What's New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server release 11.5, the system changes added to support those features, and the changes that may affect your existing applications.
- *Navigating the Documentation for Adaptive Server* – an electronic interface for using Adaptive Server. This online document provides links to the concepts and syntax in the documentation that are relevant to each task.
- *Transact-SQL User's Guide* – documents Transact-SQL, Sybase's enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the *pubs2* and *pubs3* sample databases.
- *System Administration Guide* – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources and user and system databases, and specifying character conversion, international language, and sort order settings.
- *Adaptive Server Reference Manual* – contains detailed information about all Transact-SQL commands, functions, procedures, and datatypes. This manual also contains a list of the Transact-SQL reserved words and definitions of system tables.

- *Performance and Tuning Guide* – explains how to tune Adaptive Server for maximum performance. This manual includes information about database design issues that affect performance, query optimization, how to tune Adaptive Server for very large databases, disk and cache issues, and the effects of locking and cursors on performance.
- The *Utility Programs* manual for your platform – documents the Adaptive Server utility programs, such as **isql** and **bcp**, which are executed at the operating system level.
- *Security Administration Guide* – explains how to use the security features provided by Adaptive Server to control user access to data. This manual includes information about how to add users to Adaptive Server, administer both system and user-defined roles, grant database access to users, and manage remote Adaptive Servers.
- *Security Features User's Guide* – provides instructions and guidelines for using the security options provided in Adaptive Server from the perspective of the non-administrative user.
- *Error Messages* and *Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
- *Component Integration Services User's Guide for Adaptive Server Enterprise and OmniConnect* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- *Adaptive Server Glossary* – defines technical terms used in the Adaptive Server documentation.
- *Master Index for Adaptive Server Publications* – combines the indexes of the *Adaptive Server Reference Manual, Component Integration Services User's Guide, Performance and Tuning Guide, Security Administration Guide, Security Features User's Guide, System Administration Guide,* and *Transact-SQL User's Guide*.

Other Sources of Information

Use the SyBooks™ and SyBooks-on-the-Web online resources to learn more about your product:

- SyBooks documentation is on the CD that comes with your software. The DynaText browser, also included on the CD, allows

you to access technical information about your product in an easy-to-use format.

Refer to *Installing SyBooks* in your documentation package for instructions on installing and starting SyBooks.

- SyBooks-on-the-Web is an HTML version of SyBooks that you can access using a standard Web browser.

  To use SyBooks-on-the-Web, go to http://www.sybase.com, and choose Documentation.

Conventions

The following sections describe conventions used in this manual.

Formatting SQL Statements

SQL is a free-form language: there are no rules about the number of words you can put on a line, or where you must break a line. However, for readability, all examples and syntax statements in this manual are formatted so that each clause of a statement begins on a new line. Clauses that have more than one part extend to additional lines, which are indented.

SQL Syntax Conventions

- Syntax statements (displaying the syntax and all options for a command) appear as follows:

  ```
  sp_dropdevice [device_name]
  ```

  or, for a command with more options:

  ```
  select column_name
      from table_name
      where search_conditions
  ```

  In syntax statements, command keywords are in normal font. Identifiers are in lowercase, normal font for keywords, and italics for user-supplied words.

- Examples showing the use of Transact-SQL commands are printed like this:

  ```
  select * from publishers
  ```

- Examples of output from the computer are printed like this:

```
pub_id  pub_name              city       state
------- --------------------- --------   ----
0736    New Age Books         Boston     MA
0877    Binnet & Hardley      Washington DC
1389    Algodata Infosystems  Berkeley   CA

(3 rows affected)
```

Table 1 describes the conventions for syntax statements used in this manual.

Table 1: Font and syntax conventions for this manual

Element	Example
Command names, command option names, utility names, utility flags, and other keywords are **bold**.	**select** **sp_configure**
Database names, datatypes, file names and path names are in *italics*.	*master* database
Variables, or words that stand for values that you fill in, are in *italics*.	`select` *column_name* `from` *table_name* `where` *search_conditions*
Parentheses must be typed as part of the command.	`compute` *row_aggregate* `(`*column_name*`)`
Curly braces indicate that you must choose at least one of the enclosed options. Do not type the braces.	`{cash, check, credit}`
The vertical bar separating options within curly braces indicate that you may choose only one of the enclosed options.	`{extra_cheese | avocados | sour_cream}`
Square brackets indicate that choosing one or more of the enclosed options is optional. Do not type the brackets.	`[anchovies]`
The comma separating options within square brackets indicates that you may choose as many of the enclosed options as you want. Separate your choices with commas typed as part of the command.	`[extra_cheese, avocados, sour_cream]`

Table 1: Font and syntax conventions for this manual (continued)

Element	Example
An ellipsis (...) indicates that you may repeat the last choice or group of choices as many times as you like.	`buy thing = price [cash	check	credit]` `[, thing = price [cash	check	credit]]...` In this example, you must buy at least one thing and give its price. You may choose a method of payment: one of the items enclosed in square brackets. You may also choose to buy additional things: as many of them as you want. For each thing you buy, give its name, its price, and (optionally) a method of payment.

Case

In this manual, most of the examples are in lowercase. However, you can disregard case when typing Transact-SQL keywords. For example, **SELECT**, **Select**, and **select** are the same.

Note that Adaptive Server's sensitivity to the case of database objects, such as table names, depends on the sort order installed on Adaptive Server. You can change case sensitivity for single-byte character sets by reconfiguring the Adaptive Server sort order. See Chapter 13, "Configuring Character Sets, Sort Orders, and Languages," in the *System Administration Guide* for more information.

Expressions

Several different types of expressions are used in SQL Server syntax statements.

Table 2: Types of expressions used in syntax statements

Usage	Definition
expression	Can include constants, literals, functions, column identifiers, variables, or parameters
logical expression	An expression that returns TRUE, FALSE, or UNKNOWN
constant expression	An expression that always returns the same value, such as "5+3" or "ABCDE"
float_expr	Any floating-point expression or expression that implicitly converts to a floating value

Table 2: Types of expressions used in syntax statements (continued)

Usage	Definition
integer_expr	Any integer expression, or an expression that implicitly converts to an integer value
numeric_expr	Any numeric expression that returns a single value
char_expr	An expression that returns a single character-type value
binary_expression	An expression that returns a single binary or varbinary value

If You Need Help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

1 Introduction

This chapter provides an overview of the major security features in Adaptive Server. Topics include:

Overview

Adaptive Server includes security features that enable you to protect sensitive data from inappropriate access and unauthorized disclosure. The administrators of your system determine the local application of the security policy. Your role as a user is to be aware of the security mechanisms that are available to you and to make use of them. The purpose of this manual is to describe these security mechanisms and tell you how to use them effectively.

Adaptive Server contains all of the security features included in SQL Server release 11.0.6, which was evaluated at the Class C2 security level. The requirements for the C2 criteria are given by the Department of Defense in DOD 52.00.28-STD, *Department of Defense Trusted Computer System Evaluation Criteria* (TCSEC), also known as the "Orange Book."

User Identification and Authentication

A user in Adaptive Server is given a login account with a unique ID. All of that user's activity on the server can be attributed to his or her server user ID and audited.

Adaptive Server passwords must be 6 bytes or longer. They are stored in the *master..syslogins* table in encrypted form. In addition, when you log into Adaptive Server from a client, you can choose client-side password encryption to encrypt your password before sending it over the network.

Chapter 2, "Logging into Adaptive Server," describes the login process.

Division of Roles

An important feature in Adaptive Server is the division of **roles**. Adaptive Server provides **system roles**, such as System Administrator and System Security Officer, and **user-defined roles**, which are created by a System Security Officer.

In addition, a System Security Officer can define:

- Role hierarchies – roles that contain other roles. The roles can be either user-defined or included in the system.
- Mutual exclusivity of roles – roles that are independent of each other.

More than one login account on an Adaptive Server can be granted any role, and one account can possess more than one role.

User-defined roles make it easy to establish permissions for database access for all individuals in an enterprise. When a new employee arrives, the employee can be assigned one or more roles, based upon position and data requirements. Then the employee automatically has all data access permissions that are defined for the role or roles.

System roles and user-defined roles are discussed in Chapter 3, "Roles in Adaptive Server."

Discretionary Access Controls

The discretionary access controls (DAC) are used at the discretion of object owners. They are **discretionary** because an object owner can allow you to access an object or can disallow such access.

The SQL commands **grant** and **revoke** control SQL Server's discretionary access control system. You can give various kinds of permissions to users, groups, and roles with the **grant** command and rescind them with the **revoke** command. **grant** and **revoke** are used to give users permission to create objects within a database, to create databases, and to access specified tables, views, and columns.

Some commands can be used at any time by any user with no permission required. Others can be used only by users of certain status (for example, only by a System Administrator) and are not transferable.

The ability to assign permissions for the commands that can be granted and revoked is determined by each user's status and by whether or not a user has been granted a permission with the option to grant that permission to other users.

Discretionary access controls are discussed in Chapter 4, "Granting Database Permissions."

Auditing

A principal element of a secure system is accountability. One means of ensuring this accountability is by auditing events on the system. Many events that occur in Adaptive Server can be recorded in an audit record. Each audit record can log the nature of the event, the date and time, the user responsible for it, and the success or failure of the event. Among the events that can be audited are logins and logouts, server boots, use of data access commands, attempts to access particular objects, and a particular user's actions. The **audit trail**, or log of audit records, allows the System Security Officer to reconstruct events that have occurred on the system and evaluate their impact.

Auditing is managed by the System Security Officer, and is discussed in Chapter 8, "Auditing," in the *Security Administration Guide.*

2 Logging into Adaptive Server

This chapter provides information about Adaptive Server login accounts, remote logins, and logging into and out of Adaptive Server. Topics include:

Understanding Your Adaptive Server Login Account

Each Adaptive Server user has a login account identified by a unique login name and a password. The account is established by a System Security Officer. Login accounts have the following characteristics:

- A login name, unique on that server.
- A password
- A default database (optional). If one is defined, you start each Adaptive Server session in that database without having to issue the **use** command. If none is defined, you start each session in the *master* database.
- A default language (optional). This specifies the language in which prompts and messages will be displayed to you. If one is not defined, Adaptive Server's default language is used.
- A full name (optional). This is your full name, which can be useful for documentation and identification purposes. If it is omitted, no full name is recorded for your account.

Group Membership

In Adaptive Server, groups provide a convenient way to grant and revoke permissions to more than one user at a time within a database. For example, if everyone who works in the "Sales" department needs access to certain tables, all of those users can be put into a group called "sales". The Database Owner can grant specific access permissions to that group rather than having to grant permission to each user individually.

A group is created within a database, not on the server as a whole. The Database Owner is responsible for creating groups and

assigning users to them. You are always a member of the "public" group, which includes all users on Adaptive Server. You can also belong to one other group. You can use the **sp_helpuser** system procedure to find out what group you are a member of. The syntax for **sp_helpuser** is:

```
sp_helpuser user_name
```

Role Membership

In Adaptive Server, a System Security Officer can define and create roles as a convenient way to grant and revoke permissions to several users at a time, server-wide. For example, clerical staff may need to be able to insert and select from tables in several databases, but they may not need to update them. A System Security Officer could define a role called "clerical_user_role" and grant the role to everyone in the clerical staff. Database object owners could then grant "clerical_user_role" the required privileges.

Roles can be defined in a role hierarchy, where a role such as "office_manager_role" contains the "clerical_user_role". Users who are granted roles in a hierarchy automatically have all the permissions of the roles that are lower in the hierarchy. For example, the Office Manager can perform all the actions permitted for the clerical staff. Hierarchies can include either system or user-defined roles.

You can use the following system procedures to find out more about roles assigned to you:

- **sp_displayroles**

  To find out all roles assigned to you, whether or not they are active, use **sp_displayroles.**

- **sp_activeroles**

  To find out which of your assigned roles are active, use **sp_activeroles**. If you specify the **expand_down** parameter, Adaptive Server displays any roles contained within your currently active roles.

The syntax is:

```
sp_displayroles user_name

sp_activeroles expand_down
```

For more information about roles, see Chapter 3, "Roles in Adaptive Server."

Getting Information About Your Adaptive Server Account

You can get information about your own Adaptive Server login account with the **sp_displaylogin** system procedure. The syntax is:

```
sp_displaylogin
```

Adaptive Server returns the following information:

- Your server user ID
- Your login name
- Your full name
- Any roles granted to you (regardless of whether they are currently active)
- Whether your account is locked
- The date when you last changed your password

Changing Your Password

It is a good idea to change your password periodically. The System Security Officer can configure Adaptive Server to require you to change your password at preset, regular intervals. If this is the case on your server, Adaptive Server will notify you when it is time to change your password.

➤ ***Note***

If you use remote servers, you must change your password on all remote servers that you access **before** you change it on your local server. For more information, see "Changing Your Password on a Remote Server" on page 2-5.

You can change your password at any time with the system procedure **sp_password**. The syntax is:

```
sp_password old_passwd, new_passwd
```

where *old_passwd* is the old password, and *new_passwd* is your new password.

Use the following rules when creating a new password:

- It must be at least 6 bytes long.
- It can be any printable letters, numbers, or symbols.

- The maximum size for a password is 30 bytes. If your password exceeds 30 bytes, Adaptive Server uses the first 30 characters, and then informs you that it is ignoring the rest.

When you specify a password, enclose it in quotation marks if:

- It includes characters other than A–Z, a–z, 0–9, _, #, valid single-byte or multibyte alphabetic characters, or accented alphabetic characters
- It begins with a number 0–9

The following example shows how to change the password "terrible2" to "3blindmice":

```
sp_password terrible2, "3blindmice"
```

A return status of 0 means that the password was changed. For more information about **sp_password**, see the *Adaptive Server Reference Manual*.

Protecting Your Password

Your password is the first line of defense against Adaptive Server access by unauthorized people. When you create your password, choose one that cannot be guessed, following these guidelines:

- Do not use personal information, such as your birthday, street address, or any other number that has anything to do with your personal life.
- Do not use names of pets or loved ones.
- Do not use words that appear in the dictionary or words spelled backwards.

The most difficult passwords to guess are those that combine uppercase and lowercase letters or both numbers and letters. Protecting your password is your responsibility. Never give anyone your password, and never write it down where anyone can see it.

Understanding Remote Logins

You can execute stored procedures on a remote Adaptive Server if you have been granted access to the remote server and an appropriate database on that server. Remote execution of a stored procedure is a **remote procedure call** (RPC).

To get access to a remote server, the following events must occur:

- The remote server must be known to your local server. This occurs when the System Security Officer executes **sp_addserver.**
- You must acquire a login on the remote server. This is accomplished when the System Administrator executes **sp_addremotelogin.**
- You must be added as a user to the appropriate database on the remote server. This is accomplished when the Database Owner executes **sp_adduser.**

These procedures are discussed in Chapter 7, "Managing Remote Servers," in the *Security Administration Guide.*

When you can access the remote server, you can execute a **remote procedure call (RPC)** by qualifying the stored procedure name with the name of the remote server. For example, to execute **sp_help** on the GATEWAY server, enter:

```
GATEWAY...sp_help
```

Or, to fully qualify the name of the procedure, include the name of the database containing the procedure and the name of its owner:

```
GATEWAY.sybsystemprocs.dbo.sp_help
```

➤ *Note*

Remote logins are not included in the **evaluated configuration**.

Changing Your Password on a Remote Server

You must change your password on all remote servers that you access **before** you change it on your local server because of the way password checking is done. If you change your password on the local server first, when you issue the remote procedure call (RPC) to execute **sp_password** on the remote server, your local and remote passwords will not match and the command will fail.

The syntax for changing your password on the remote server is:

```
remote_server...sp_password old_passwd, new_passwd
```

For example:

```
GATEWAY...sp_password terrible2, "3blindmice"
```

For information on changing your password on the local server, see "Changing Your Password" on page 2-3.

Logging Into Adaptive Server

You can log into Adaptive Server using any of the following methods:

- Use Sybase Central™, as discussed in "Connecting to Adaptive Server with Sybase Central" on page 2-6.
- Use **isql**, as discussed in "Logging In with isql" on page 2-6.
- Use Data Workbench®, as discussed in "Logging In via Data Workbench" on page 2-7.
- Through an application written using Open Client Client-Library or Open Client DB-Library™. This is discussed in "Handling Client / Server Connection Security with Open Client" on page 2-8.

➤ ***Note***

In addition to using these methods for invoking Adaptive Server, you can use utilities such as **bcp** and **defncopy** directly from the operating system. You can use **bcp** to import and export data to and from Adaptive Server and use **defncopy** to copy the definitions of objects such as procedures and defaults to and from Adaptive Server. For information about how to use these utilities and for more information about **isql**, see *Utility Programs* manual for your platform

Connecting to Adaptive Server with Sybase Central

Sybase Central provides a graphical user interface for connecting to Sybase products such as Adaptive Server. From the Sybase Central menu, choose Tools, and then choose Connect. Then, supply your login and password in the dialog box.

Logging In with *isql*

You can log into Adaptive Server directly from the operating system with the **isql** utility program.

To use **isql**, type this at your operating system prompt:

```
isql
```

You will then see this prompt:

```
Password:
```

Type your password at the prompt, and then press the Return key. The password is not shown on the screen as you type. Note that login names and passwords are case sensitive. The following prompt apprears:

```
1>
```

At this point, you can start issuing Transact-SQL commands.

When you specify **isql** without a login name, Adaptive Server assumes that your login name is the same as your operating system name. For details about specifying your server login name and other parameters for **isql**, see *Utility Programs* manual for your platform.

➤ ***Note***

Do not use the **-P** option to **isql** to specify your password, because another user might see your password.

Logging Out of *isql*

You can log out of **isql** at any time by typing:

```
quit
```

or

```
exit
```

Either of these commands returns you to the operating system prompt.

Logging In via Data Workbench

You can connect to Adaptive Server via Data Workbench.

To start Data Workbench, at the operating system prompt, enter:

```
dwb
```

A login window opens, which prompts you for your Adaptive Server password. For more information about Data Workbench, see the *Data Workbench User's Guide.*

Logging Out of Data Workbench

To exit Data Workbench, return to the main window, select **/** and then **exit.**

Handling Client/Server Connection Security with Open Client

Open Client Client-Library and Open Client DB-Library include routines that handle the encryption of passwords between the client and the server.

Login Security in DB-Library

If you are using DB-Library to write your application, the following routines handle login security:

- **DBSETLENCRYPT** specifies whether network password encryption is used when logging into SQL Server release 10.0 or logging into Adaptive Server.
- **dbsechandle** installs user-defined functions to handle encrypted passwords. This is most useful for gateway applications.

These routines are discussed in detail in the *Open Client DB-Library/C Reference Manual.*

➤ ***Note***

Open Client DB-Library/C is not included in the **evaluated configuration**.

Login Security in Client-Library

Client-Library automatically handles password encryption between client and server, unless the application is a gateway. For gateway applications, you must handle password encryption explicitly, passing the server's encryption key on to the client and then returning the encrypted password back to the server.

For more information about these topics, see the *Open Client Client-Library/C Reference Manual.*

3 Roles in Adaptive Server

This chapter provides information about the use of roles in Adaptive Server. Topics include:

Understanding Roles

An important feature in Adaptive Server is the division of **roles**. Adaptive Server provides **system roles**, such as System Administrator and System Security Officer, and **user-defined roles**, which are created by a System Security Officer.

In addition, a System Security Officer can define:

- Role hierarchies – roles that contain other roles. The roles can be either user-defined or system roles.
- Mutual exclusivity of roles – roles that are independent of each other.

More than one login account on an Adaptive Server can be granted any role, and one account can possess more than one role.

System Roles

Various security-related, administrative, and operational tasks are grouped into the following roles:

- **oper_role** for an Operator, who can back up and load databases server-wide.
- **sa_role** for a System Administrator, whose tasks include managing and maintaining Adaptive Server databases and disk storage
- **sso_role** for a System Security Officer, who performs security-related tasks

User-Defined Roles

Adaptive Server enables a System Security Officer to create roles and grant them to users, groups of users, or other roles. For example, a System Security Officer might create a “financial_analyst” role and grant that role to users that perform financial analyst tasks. Similarly, a role “salary_administrator” might be created and assigned to those who handle payroll for your organization. Object owners can grant database access as appropriate to each role.

Role Hierarchies and Mutual Exclusivity

A System Security Officer can define role hierarchies such that if a user has one role, the user automatically has roles lower in the hierarchy. For example, the “chief_financial_officer” role might contain both the “financial_analyst” and the “salary_administrator” roles, as shown in Figure 3-1:

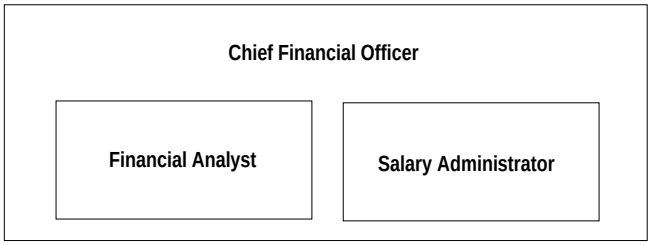

Figure 3-1: Role hierarchy

The Chief Financial Officer can perform all tasks and see all data that can be viewed by the Salary Administrator and Financial Analyst.

Two roles can be defined to be mutually exclusive for:

- Membership – one user cannot be granted both roles. For example, an installation might not want the “payment_requestor” and “payment_approver” roles to be granted to the same user.
- Activation – one user cannot activate, or enable, both roles. For example, a user might be granted both the “senior_auditor” and the “equipment_buyer” roles, but the installation may not want to permit the user to have both roles enabled at the same time.

System roles and user-defined roles can both be defined to be in a role hierarchy or to be mutually exclusive. For example, your

installation might want a “super_user” role to contain the System Administrator, Operator, and “tech_support” roles. In addition, your installation might want to define the System Administrator and System Security Officer roles to be mutually exclusive for membership; that is, one user cannot be granted both roles.

System and Security Administration Roles

The System Administrator, System Security Officer, and Operator roles are essential for managing Adaptive Server.

The System Administrator

A **System Administrator** performs administrative tasks that are unrelated to specific applications. The System Administrator is not necessarily one individual; the role can be granted to any number of individual login accounts. In a large organization, the System Administrator’s role may be carried out by several people or groups.

System Administrator Tasks

System Administrator tasks include:

- Installing Adaptive Server
- Managing disk storage
- Granting permissions to Adaptive Server users
- Modifying, dropping, and locking server login accounts
- Monitoring Adaptive Server’s automatic recovery procedure
- Diagnosing system problems and reporting them as appropriate
- Fine-tuning Adaptive Server by changing configurable system parameters
- Creating user databases and granting ownership of them
- Granting and revoking the System Administrator role
- Setting up groups (which is convenient for granting and revoking permissions)

System Administrator Permissions

Adaptive Server does no discretionary access control (DAC) permission checking when a System Administrator accesses objects. A System Administrator takes on the identity of Database Owner in any database he or she enters, including *master*.

There are several commands and system procedures that only a System Administrator can issue and on which permissions cannot be transferred to other users.

System Security Officer

The **System Security Officer** is responsible for security-sensitive tasks in Adaptive Server, such as:

- Creating server login accounts
- Granting and revoking the System Security Officer and Operator roles
- Creating and granting user-defined roles.
- Changing the password of any account
- Setting the system-wide password expiration interval
- Managing the audit system

The System Security Officer can access any database but has no special privileges on objects within databases, with the exception of the *sybsecurity* database. Only a System Security Officer can access the *sybsecurity* database and its tables. There are also several system procedures that only a System Security Officer can execute and on which permissions cannot be transferred to other users.

Operator

An **Operator** is a user who can back up and load databases on a server-wide basis. The Operator role allows a single user to use the **dump database, dump transaction, load database**, and **load transaction** commands to back up and restore all databases on a server without having to be the owner of each database. These operations can be performed in a single database by the Database Owner and the System Administrator.

Data Ownership Roles

There are two types of owners recognized by Adaptive Server:

- Database Owner
- Database object owner

Database Owner

The **Database Owner** is the creator of a database or someone to whom database ownership has been transferred by the System Administrator. The System Administrator grants users the authority to create databases with the **grant** command.

A Database Owner logs into Adaptive Server using his or her assigned login name and password. In other databases, that owner is known by his or her database user name; in his or her own database, Adaptive Server recognizes the user as "dbo".

Database Owner Tasks

The owner of a database can:

- Run the system procedure **sp_adduser** to allow other Adaptive Server users access to the database
- Use the **grant** command to give other users permission to create objects and execute commands within the database
- Create and drop user messages within the database

The Database Owner can access any object inside the database that he or she owns.

Database Object Owner

Database objects are tables, indexes, views, defaults, triggers, rules, constraints, and procedures. A user who creates a database object is its owner. The Database Owner must first grant the user permission to create objects of a particular type.

Database Object Owner Tasks

The database object owner creates an object using the appropriate **create** statement, and then grants permission to other users.

Database Object Owner Permissions

The creator of a database object is automatically granted all permissions on that object. A System Administrator also has all permissions on the object, as long as he or she qualifies the object name with the owner's name. The owner of an object must explicitly grant permissions to other users before they can access the object. Even the Database Owner cannot use an object directly unless the object owner grants him or her the appropriate permission. However, the Database Owner can always use the **setuser** command to impersonate any other user in the database, including the object owner.

When a database object is owned by someone other than the Database Owner, a user other than the owner (including a System Administrator) must qualify the name of that object with the object owner's name—*ownername.objectname*—to access the object. If an object or a procedure needs to be accessed by a large number of users, particularly in ad hoc queries, having these objects owned by "dbo" greatly simplifies access.

Creating User-Defined Roles

User-defined roles are created and granted by a System Security Officer. The System Security Officer might use a naming convention such as appending "_role" to a role name. For example, the name of the role for a writer might be "writer_role". User-defined roles can be granted only by the System Security Officer and cannot be granted with the **with grant option**.

A user who has been granted a user-defined role or roles must activate each granted role individually using **set role**. The syntax is:

```
set role role_name
```

If the role has an attached password, use the following syntax:

```
set role role_name with passwd "password" [on|off]
```

To set up your roles to be activated by default when you log into Adaptive Server, use the system procedure **sp_modifylogin** with the following syntax:

```
sp_modifylogin loginname [,"add default role" |"drop
   default role"] role_name
```

The **add default role** parameter includes the named role in the roles that activate by default at login.

The **drop default role** parameter drops the named role from the roles that activate by default at login.

Displaying Information About Roles

Table 3-1 lists the system procedures and functions to use to find information about roles and the section in this chapter that provides details.

Table 3-1: Finding information about roles

To Display Information About	Use	See
The role ID of a role name	**role_id** system function	"Finding a Role ID" on page 3-7
The role name of a role ID	**role_name** system function	"Finding a Role Name" on page 3-8
System roles	**show_role** system function	"Viewing Information About System Roles" on page 3-8
Role hierarchies and roles that have been granted to a user or users	**sp_displayroles** system procedure	"Displaying a Role Hierarchy" on page 3-8
Whether one role contains another role in a role hierarchy	**role_contain** system function	"Viewing User Roles in a Hierarchy" on page 3-9
Whether two roles are mutually exclusive	**mut_excl_roles** system function	"Determining Mutual Exclusivity" on page 3-9
Roles that are active for the current session	**sp_activeroles** system procedure	"Determining Role Activation" on page 3-9
Whether you have activated the correct role to execute a procedure	**proc_role** system function	"Checking for Roles in Stored Procedures" on page 3-9
Logins, including roles that have been granted	**sp_displaylogin** system procedure	"Displaying Login Account Information" on page 3-10
Permissions for a user, group, or role	**sp_helprotect** system procedure	"Checking Permissions" on page 3-11

Finding a Role ID

To find the **role_id** of a role when you know the role name, use the **role_id** system function. The syntax is:

```
role_id(role_name)
```

Any user can execute **role_id**. If the role is a valid one, **role_id** returns the server-wide ID of the role (*srid*). The *syssrvroles* system table

contains an *srid* column with the role ID and a *name* column with the role name. If the role is not valid, **role_id** returns NULL.

Finding a Role Name

To find the **role_name** of a role when you know the role id, use the **role_name** system function. The syntax is:

```
role_name(role_id)
```

Any user can execute **role_name**.

Viewing Information About System Roles

The **show_role** system function shows the currently active **system roles** for the specified login. The syntax is:

```
show_role()
```

If you have not activated any system role, **show_role** returns NULL. If you are a Database Owner, and you execute **show_role** after using **setuser** to impersonate another user, **show_role** returns your own active system roles, not the active system roles of the user you are impersonating.

Any user can execute **show_role**.

➤ *Note*

The **show_role** function does not give information about user-defined roles.

Displaying a Role Hierarchy

You can see all roles granted to your login name or see the entire hierarchy tree of roles displayed in table format using **sp_displayroles**. The syntax is:

```
sp_displayroles {login_name | rolename [, expand_up |
   expand_down]}
```

Any user can execute **sp_displayroles** to see their own roles. Only the System Security Officer or the System Administrator can view information about roles granted to other users.

Viewing User Roles in a Hierarchy

The **role_contain** built-in system function shows whether any one role you specify contains any other role you specify. The syntax is:

```
role_contain (["role1", "role2"])
```

If *role1* contains *role2*, **role_contain** returns 1.

Any user can execute **role_contain**.

Determining Mutual Exclusivity

You can use the **mut_excl_roles** function to determine whether any two roles assigned to you are mutually exclusive and the level at which they are mutually exclusive. The syntax is:

```
mut_excl (role1, role2) [membership | activation]
```

Any user can execute **mut_excl_roles**. If the specified roles, or any role contained by either specified role, are mutually exclusive, **mut_excl_roles** returns 1; if the roles are not mutually exclusive, **mut_excl_roles** returns 0.

Determining Role Activation

To find all active roles for the current login session of Adaptive Server, use **sp_activeroles**. The syntax is:

```
sp_activeroles [expand_down]
```

If you specify **expand_down**, **sp_activeroles** displays the hierarchy of all roles contained by any roles granted to you.

Any user can execute **sp_activeroles**.

Checking for Roles in Stored Procedures

Use the **proc_role** function within a stored procedure to guarantee that only users with a specific role can execute the procedure. Only **proc_role** provides a fail-safe way to prevent inappropriate access to a particular stored procedure.

You can use the **grant execute** command to grant execute permission on a stored procedure to all users who have been granted a specified role. Similarly, **revoke execute** removes this permission.

However, **grant execute** permission does not prevent users who do not have the specified role from being granted execute permission on a stored procedure. If you want to ensure, for example, that all users who are not System Administrators can never be granted permission to execute a stored procedure, you can use the **proc_role** system function within the stored procedure itself. It checks to see whether the invoking user has the correct role to execute the procedure.

proc_role takes a string for the required role and returns 1 if the invoker possesses it. Otherwise, it returns 0.

For example, here is a procedure that uses **proc_role** to see if the user has the **sa_role** role:

```
create proc test_proc
as
if (proc_role("sa_role") = 0)
begin
    print "You don't have the right role"
    return -1
end
else
    print "You have SA role"
    return 0
```

Displaying Login Account Information

You can use the **sp_displaylogin** system procedure to display information about a login account, including any roles granted to that account. The syntax is:

sp_displaylogin [*loginame*]

If you are not a System Security Officer or System Administrator, you can get information only about your own account, and you do not need to use the *loginame* parameter. **sp_displaylogin** displays your server user ID, login name, full name, roles, date of last password change, and whether your account is locked.

If you are a System Security Officer or System Administrator, you can use the *loginame* parameter to access information about any login.

Checking Permissions

To check on the permissions granted to users, groups, or roles, or permissions granted on database objects, use the **sp_helprotect** system procedure. The syntax is:

```
sp_helprotect [name [,username [,"grant"
   [,"none"|"granted"|"enabled"|role_name]]]]
```

Any user can use **sp_helprotect** to view his or her own permissions. Depending on the parameter you specify, **sp_helprotect** displays all permissions granted a user or role, permissions on any specified database object, permissions granted to active roles, and whether the permissions can be granted to others. The display always includes information on permissions granted to the group of which the specified user is a member.

If you do not specify any parameters, **sp_helprotect** returns permission information on all your active roles.

Only the System Security Officer or System Administrator can use **sp_helprotect** to display the permissions granted to other users.

4 Granting Database Permissions

This chapter discusses **discretionary access controls**, or **permissions**, which are used at the discretion of an object's owner. Discretionary access permissions are maintained by using the **grant** and **revoke** commands. This chapter provides database object owners with the information they need to grant and revoke privileges on their objects to other users. Topics include:

Assigning Permissions to Users

The SQL commands **grant** and **revoke** control Adaptive Server's command and object protection system. You can give various kinds of permissions to users, groups, and roles by using the **grant** command. You rescind these permissions by using the **revoke** command. **grant** and **revoke** are used to control permission to:

- Create databases
- Create objects within a database
- Access tables, views, and columns
- Execute stored procedures

The ability to assign permissions for the commands that can be granted and revoked is determined by:

- A user's status as System Administrator, Database Owner, or database object owner
- Whether a user has been granted a permission with the option to grant that permission to other users.

The Database Owner does not automatically receive permissions on objects owned by other users. However, a Database Owner or System Administrator can acquire any permission by using the **setuser** command to assume the identity of the object owner, and then writing the appropriate **grant** or **revoke** statements.

You can use **grant** and **revoke** to assign two kinds of database permissions: **object access permissions** and **object creation permissions**. These topics are discussed in "Understanding Object Access Permissions" on page 4-2 and in "Understanding Object Creation Permissions" on page 4-7.

For more information about permissions, see Chapter 5, "Managing User Permissions" in the *Security Administration Guide*, and **grant** and **revoke** in the *Adaptive Server Reference Manual*.

Understanding Object Access Permissions

Object access permissions regulate the use of certain commands that access certain database objects. For example, you must explicitly be granted permission to use the **select** command on the *authors* table. Object permissions default to both the System Administrator and the object's owner, and can be granted to other users. Object access permissions are granted and revoked by the owner of the object, who can grant them to other users.

Table 4-1 lists the types of object access permissions and the objects to which they apply:

Table 4-1: Permissions and the objects to which they apply

Permission	Object
select	Table, view, column
update	Table, view, column
insert	Table, view
delete	Table, view
references	Table, column
execute	Stored procedure

Granting and Revoking Object Access Permissions

You use the **grant** command to grant object access permissions and the **revoke** command to revoke them.

The syntax is:

```
grant {all | permission_list}
   on {table_name [(column_list)] |
      view_name [(column_list)] |
      stored_procedure_name}
   to {public | name_list | role_name}
   [with grant option]

revoke [grant option for]
   {all | permission_list}
   on {table_name [(column_list)] |
      view_name [(column_list)] |
      stored_procedure_name}
   from {public | name_list | role_name}
   [cascade]
```

The following rules apply:

- You can include more than one command in *permission_list*. Separate the commands with commas.
- If you use the keyword **all** in the statements for object access permissions, all the permissions applicable to the object are granted or revoked.
- The contents of *permission_list* varies according to the type of object on which you are granting permissions:
  - When you use the **grant** or **revoke** command to assign permissions on a table or view, *permission_list* can consist of any combination of **select**, **insert**, **delete**, **references**, and **update** statements.
  - When you grant permissions on columns in a table, *permission_list* can include **select** or **update** or **references**, or all three. To use **select** *, you must have **select** permission on all columns in the table.
  - When you are granting permissions on stored procedures, *permission_list* can include **execute** only.
- The **on** clause specifies the object on which you are granting or revoking permission. You can grant or revoke privileges on tables, views, and stored procedures for only one object at a time. *table_name*, *view_name*, or *stored_procedure_name* is the name of the table, view, or stored procedure. You can grant privileges for more than one column at a time, but all the columns must be in the same table or view. *column_list* is a list of columns separated by commas. For example, to specify the *price* and *total_sales* columns of the *titles* table, enter:

```
on titles(price, total_sales)
```

- Use **with grant option** to allow a specified user or users to grant permissions to other users.
- The **grant option for** clause of the **revoke** command revokes grant permission from the specified user or users. If the user has granted permissions to other users, you must use the **cascade** option as well to revoke permissions from those users.
- The keyword **public** refers to the "public" group, which includes all users of Adaptive Server. **public** means slightly different things for **grant** and **revoke**:
  - For **grant**, **public** includes you, the object owner. Therefore, if you have revoked permissions from yourself on your object, and later you **grant** permissions to **public**, you regain the permissions along with the rest of "public".
  - For **revoke** on object access permissions, **public** excludes the owner.
- The *name_list* includes the names of:
  - Groups
  - Users
  - A combination of users and groups with each name separated from the next by a comma
- The *role_name* is the name of an Adaptive Server role. This allows you to grant permissions to all users who have been granted a specific role. The role can be a system role such as **sa_role** (System Administrator), **sso_role** (System Security Officer), and **oper_role** (Operator) or a user-defined role such as **financial_analyst**. A System Security Officer must have already created any user-defined role you specify.
- The **with grant option** clause in a **grant** statement specifies that the user(s) specified in *name_list* can grant the specified object access permission(s) to other users. If a user has **with grant option** permission on an object, that permission is not revoked when permissions on the object are revoked from **public** or from a group to which the user belongs. User-defined roles are not granted with the **with grant option**.
- The **cascade** option in a **revoke** statement removes the specified object access permissions from the user(s) specified in *name_list* and also from any users they granted those permissions to.

Special Requirements for Compliance with the SQL92 Standard

When you have used the **set** command to turn **ansi_permissions** on, additional permissions are required for **update** and **delete** statements. The following table summarizes the required permissions.

Table 4-2: ANSI permissions for update and delete

	Permissions Required: *set ansi_permissions off*	Permissions Required: *set ansi_permissions on*
update	**update** permission on columns where values are being set	**update** permission on columns where values are being set and **select** permission on all columns appearing in the **where** clause **select** permission on all columns on the right side of the **set** clause
delete	**delete** permission on the table	**delete** permission on the table from which rows are being deleted and **select** permission on all columns appearing in the **where** clause

If **ansi permissions** is **on**, and you attempt to update or delete without having all the additional **select** permissions, the transaction is rolled back, and you receive an error message. If this occurs, the object owner must grant you **select** permission on all relevant columns.

Examples of Granting Object Access Permissions

The following statement gives Mary and the "sales" group permission to **insert** into and **delete** from the *titles* table:

```
grant insert, delete
on titles
to mary, sales
```

The following statement gives Harold permission to use the stored procedure **makelist**:

```
grant execute
on makelist
to harold
```

The following statement grants permission to execute the stored procedure **sa_only_proc** to users who have been granted the System Administrator role:

```
grant execute
on sa_only_proc
to sa_role
```

The following statement gives Aubrey permission to **select**, **update**, and **delete** from the *authors* table and to grant the same permissions to other users:

```
grant select, update, delete
on authors
to aubrey
with grant option
```

Examples of Revoking Object Access Permissions

Both of the following statements revoke permission from all users except the table owner to **update** the *price* and *total_sales* columns of the *titles* table:

```
revoke update
on titles (price, total_sales)
from public

revoke update(price, total_sales)
on titles
from public
```

The following statement revokes permission from Clare to update the *authors* table and simultaneously revokes that permission from all users to whom she had granted that permission:

```
revoke update
on authors
from clare
cascade
```

The following statement revokes permission from Operators to execute the stored procedure **new_sproc**:

```
revoke execute
on new_sproc
from oper_role
```

Understanding Object Creation Permissions

Object creation permissions regulate the use of commands that create objects. These permissions can be granted only by a System Administrator or a Database Owner.

The commands to which the object creation permissions apply are:

create database
create default
create procedure
create rule
create table
create view

Each database has its own independent protection system. In other words, being granted permission to use a certain command in one database has no effect in other databases.

Granting and Revoking Object Creation Permissions

You use the **grant** command to grant object creation permissions and the **revoke** command to revoke them. The syntax is:

```
grant {all | command_list}
   to {public | name_list | role_name}

revoke {all | command_list}
   from {public | name_list | role_name}
```

The following rules apply:

- The *command_list* is a list of the object creation permissions that you are granting or revoking. If more than one command is listed, separate them with commas. The command list can include **create database**, **create default**, **create procedure**, **create rule**, **create table**, and **create view**. **create database** permission can be granted only by a System Administrator, and only from within the *master* database.
- **all** can be used only by a System Administrator or the Database Owner. When used by a System Administrator in the *master* database, **grant all** assigns all **create** permissions, including **create database**. If the System Administrator executes **grant all** from another database, all **create** permissions are granted except **create database**. When a Database Owner who is not a System Administrator uses **grant all**, Adaptive Server grants all **create** permissions except **create database** and prints an informational message.

- The keyword **public** refers to the group "public", which includes all users of Adaptive Server. **public** means slightly different things for **grant** and **revoke**:
  - For **grant**, **public** includes the Database Owner. Therefore, if you have revoked permissions from yourself for a command, and later you **grant** permissions to **public**, you regain the permissions along with the rest of "public".
  - For **revoke**, **public** excludes the Database Owner.
- The *name_list* is a list of the names of:
  - Groups
  - Users
  - A combination of users and groups with each name separated from the next by a comma
- The *role_name* is the name of an Adaptive Server role. This allows you to grant permissions to all users who have been granted a specific role. The roles are **sa_role** (System Administrator), **sso_role** (System Security Officer), and **oper_role** (Operator).

Examples of Granting Object Creation Permissions

The following example grants permission to Mary, Jane, and Bob to create tables and views in the current database:

```
grant create table, create view
to mary, jane, bob
```

The next example grants permission to the "admin" group to create stored procedures:

```
grant create procedure
to admin
```

Example of Revoking Object Creation Permissions

The following example revokes permission from Mary to create tables and rules in the current database:

```
revoke create table, create rule
from mary
```

Understanding *grant* and Roles

You can use the **grant** command to grant permission on objects to all users who have been granted a specified role, whether system or user-defined. This allows you to restrict use of an object to users who have been granted any of the following roles:

- System Administrator
- System Security Officer
- Operator
- Any user-defined role

You can also use the **grant** command to grant a role to a user, another role or roles, or a group. For specific information on how to grant or revoke a role, see Chapter 4, "Granting and Revoking Roles," in the *Adaptive Server Enterprise Security Administration Guide*. Only a System Security Officer can grant or revoke roles.

However, **grant** permission does not prevent users who do **not** have the specified role from being granted execute permission on a stored procedure. If you want to ensure, for example, that only System Administrators can successfully execute a stored procedure, you can use the **proc_role** system function within the stored procedure itself. It checks to see whether the invoking user has the correct role to execute the procedure. See "Displaying Information About Roles" on page 3-7 for more information.

Permissions granted to roles override permissions granted to users or groups. For example, assume John has been granted the System Security Officer role, and **sso_role** has been granted permission on the *sales* table. If John's individual permission on *sales* is revoked, he is still able to access *sales* when he has **sso_role** active because his role permissions override his individual permissions.

In granting permissions, a System Administrator is treated as the object owner. If a System Administrator grants permission on another user's object, the owner's name appears as the grantor in *sysprotects* and in **sp_helprotect** output.

If several users grant access to an object to a particular user, the user's access remains until access is revoked by all those who granted access or until a System Administrator revokes the access. If a System Administrator revokes access, the user is denied access, even though other users have granted access.

Combining *grant* and *revoke* Statements

grant and **revoke** statements are order sensitive: in case of a conflict, the most recently issued statement supersedes all others.

There are two basic styles of setting up permissions in a database or on a database object. The most straightforward style is to assign specific permissions to specific users. However, if most users are going to be granted most privileges, it's easier to assign all permissions to all users and then revoke specific permissions from specific users.

For example, a Database Owner can grant all permissions on the *titles* table to all users by issuing the following statement:

```
grant all
on titles
to public
```

Then the Database Owner can issue a series of **revoke** statements, for example:

```
revoke update
on titles (price, advance)
from public

revoke delete
on titles
from mary, sales, john
```

➤ ***Note***

Under SQL rules, the **grant** command must be used before the **revoke** command, but the two commands cannot be used within the same transaction. Therefore, when you grant "public" access to objects, and then revoke that access from an individual, there is a short period of time during which the individual has access to the objects in question. To prevent this situation, use the **create schema** command to include the **grant** and **revoke** clauses within one transaction.

Understanding Permission Order and Hierarchy

grant and **revoke** statements are sensitive to the order in which they are issued. For example, if Jose's group has been granted **select** permission on the *titles* table and then Jose's permission to **select** the *advance* column has been revoked, Jose can **select** all the columns

except *advance*, while the other users in his group can still **select** all the columns.

A **grant** or **revoke** statement that applies to a group or role changes any conflicting permissions that have been assigned to any member of that group or role. For example, if the owner of the *titles* table has granted different permissions to various members of the *sales* group, and wants to standardize, he or she might issue the following statements:

```
revoke all on titles from sales
grant select on titles(title, title_id, type,
        pub_id)
    to sales
```

Similarly, a **grant** or **revoke** statement issued to **public** will change, for all users, all previously issued permissions that conflict with the new regime.

The same **grant** and **revoke** statements issued in different orders can create entirely different situations. For example, the following set of statements leaves Jose, who belongs to the **public** group, without any **select** permission on *titles*:

```
grant select on titles(title_id, title) to jose
revoke select on titles from public
```

In contrast, the same statements issued in the opposite order result in only Jose having **select** permission and only on the *title_id* and *title* columns:

```
revoke select on titles from public
grant select on titles(title_id, title) to jose
```

When you use the keyword **public** with **grant**, you are including yourself. With **revoke** on object creation permissions, you are included in **public** unless you are the Database Owner. With **revoke** on object access permissions, you are included in **public** unless you are the object owner. You may want to deny yourself permission to use your own table, while giving yourself permission to access a view built on it. To do this you must issue **grant** and **revoke** statements explicitly setting your permissions. You can reinstitute the permission with a **grant** statement.

Getting Information About Permissions

Table 4-3 lists the system procedures to use to report permissions information and the section in this chapter that provides details.

Table 4-3: Finding information about permissions

To Find Permissions Information On	Use	See
Database objects or users	**sp_helprotect**	"Reporting Information on Objects or Users" on page 4-12
Specific tables	**sp_table_privileges**	"Reporting Information on Specified Tables" on page 4-14
Specific columns in a table	**sp_column_privileges**	"Reporting Information On Specified Columns" on page 4-14

Reporting Information on Objects or Users

The system procedure **sp_helprotect** reports on permissions by database object or by user, and (optionally) by user for a specified object. Any user can execute this procedure. The syntax is:

```
sp_helprotect name [, username [, "grant"
   [,"granted"|,"none"|"enabled"|rolename ]]]
```

The following rules apply:

- The *name* is either the name of the table, view, or stored procedure; or the name of a user, group, or role in the current database.
- If you specify *username*, only that user's permissions on the specified object are reported. If *name* is not an object, **sp_helprotect** checks whether *name* is a user, group, or role. If it is, the permissions for the user, group, or role are listed.
- If you specify **grant**, and *name* is not an object, **sp_helprotect** displays all permissions granted by **with grant option.**
- If you specify:
  - **granted** – Adaptive Server takes into account all roles granted to you when calculating permissions.
  - **none** – Adaptive Server does not take into account any roles granted to you when calculating permissions.
  - **enabled** – Adaptive Server takes into account any of your roles that you have activated when calculating permissions.

- **rolename** – Adaptive Server takes only the specified role into account when calculating permissions.

For example, suppose you issue the following series of **grant** and **revoke** statements:

```
grant select on titles to judy
grant update on titles to judy
revoke update on titles(contract) from judy
grant select on publishers to judy
    with grant option
```

To determine the permissions Judy now has on each column in the *titles* table, enter:

```
sp_helprotect titles, judy
```

The following results are displayed:

```
grantor grantee type   action   object     column      grantable
------- ------ -----   ------   ------     ------      -------
dbo     judy   Grant   Select   titles     All         FALSE
dbo     judy   Grant   Update   titles     advance     FALSE
dbo     judy   Grant   Update   titles     notes       FALSE
dbo     judy   Grant   Update   titles     price       FALSE
dbo     judy   Grant   Update   titles     pub_id      FALSE
dbo     judy   Grant   Update   titles     pubdate     FALSE
dbo     judy   Grant   Update   titles     title       FALSE
dbo     judy   Grant   Update   titles     title_id    FALSE
dbo     judy   Grant   Update   titles     total_sales FALSE
dbo     judy   Grant   Update   titles     type        FALSE
```

The first line of the results shows that the Database Owner ("dbo") gave Judy permission to select all columns of the *titles* table. The remaining lines indicate that she can update only the columns listed under *column*. The "FALSE" values in the *grantable* column indicate that Judy cannot grant **select** or **update** permissions to any other user.

To see Judy's permissions on the *publishers* table, enter:

```
sp_helprotect publishers, judy
```

In the following results, the value in the *grantable* column is "TRUE", meaning that Judy can grant the permission to other users.

```
grantor grantee type   action   object     column      grantable
------- ------ -----   ------   ------     ------      -------
dbo     judy   Grant   Select   publishers all         TRUE
```

Reporting Information on Specified Tables

The catalog stored procedure **sp_table_privileges** returns permissions information about a specified table. The syntax is:

```
sp_table_privileges table_name [, table_owner
   [, table_qualifier]]
```

The parameters are used as follows:

- *table_name* is the name of the table.
- *table_owner* can be used to specify the name of the table owner, if it is not "dbo" or the user executing **sp_column_privileges.**
- *table_qualifier* is the name of the current database.
- Use **null** for parameters that you do not want to specify.

For example, the following statement returns information about all permissions granted on the *titles* table:

```
sp_table_privileges titles
```

For more information, see **sp_table_privileges** in the *Adaptive Server Reference Manual*.

Reporting Information On Specified Columns

The catalog stored procedure **sp_column_privileges** returns information about permissions on columns in a table. The syntax is:

```
sp_column_privileges table_name [, table_owner
   [, table_qualifier [, column_name]]]
```

The following rules apply:

- *table_name* is the name of the table.
- *table_owner* can be used to specify the name of the table owner, if it is not "dbo" or the user executing **sp_column_privileges.**
- *table_qualifier* is the name of the current database.
- *column_name* is the name of the column on which you want to see permissions information.
- Use **null** for parameters that you do not want to specify.

For example, the following statement:

```
sp_column_privileges publishers, null, null, pub_id
```

returns information about the *pub_id* column of the *publishers* table. For more information, see **sp_column_privileges** in the *Adaptive Server Reference Manual*.

Granting Permissions on Views and Stored Procedures

Views and stored procedures can serve as security mechanisms. A user can be granted permission on a view or on a stored procedure even if he or she has no permissions on objects that the view or procedure references. For this to be possible, the view or stored procedure and its underlying objects must be owned by the same user. Otherwise, the person using the stored procedure or view must be granted object access permissions on the underlying objects as well as on the view or stored procedure.

Adaptive Server makes permission checks, as required, when the view or procedure is used. When you create the view or procedure, Adaptive Server makes no permission checks on the underlying objects.

Using Views As Security Mechanisms

Through a view, users can query and modify only the data defined by the view. The rest of the database is neither visible nor accessible. Data in an underlying table that is not included in the view is hidden from users who are authorized to access the view but not the underlying table.

Permission to access the view must be explicitly granted or revoked, regardless of the set of permissions in force on the view's underlying tables. By defining different views and selectively granting permissions on them, a user (or any combination of users) can be restricted to different subsets of data. The following examples illustrate the use of views for security purposes:

- Access can be restricted to a subset of the rows of a base table (a value-dependent subset). For example, you might define a view that contains only the rows for business and psychology books, in order to keep information about other types of books hidden from some users.
- Access can be restricted to a subset of the columns of a base table (a value-independent subset). For example, you might define a view that contains all the rows of the *titles* table, but omits the *price* and *advance* columns, since this information is sensitive.

- Access can be restricted to a row-and-column subset of a base table.
- Access can be restricted to the rows that qualify for a join of more than one base table. For example, you might define a view that joins the *titles*, *authors*, and *titleauthor* tables in order to display the names of the authors and the books they have written. This view would hide personal data about authors and financial information about the books.
- Access can be restricted to a statistical summary of data in a base table. For example, you might define a view that contains only the average price of each type of book.
- Access can be restricted to a subset of another view, or of some combination of views and base tables.

Example of Using Views As Security Mechanisms

As an example, suppose you want to prevent some users from accessing the columns in the *titles* table that have to do with money and sales. You could:

- Create a view of the *titles* table that omits those columns
- Revoke permission from "public" to access the table
- Give all users permission on the view but only the Sales department permission on the table

To accomplish these tasks, you would use the following commands:

```
grant all on bookview to public

revoke all on titles from public

grant all on titles to sales
```

To set up these privilege conditions without using a view you could enter the following commands:

```
grant all on titles to public

revoke select, update on titles (price, advance,
   total_sales)
from public

grant select, update on titles (price, advance,
   total_sales)
to sales
```

One possible problem with the second solution is that users not in the *sales* group who enter the command:

```
select * from titles
```

might be surprised to see the message that includes the phrase:

```
permission denied
```

Adaptive Server expands the asterisk into a list of all the columns in the *titles* table, and because permission on some of these columns has been revoked from non-sales users, Adaptive Server returns an error message. The message lists the columns for which the user does not have access.

To see all the columns for which they do have permission, the non-sales users would have to name them explicitly. For this reason, creating a view and granting the appropriate permissions on it is a better solution.

Using Views for Context-Sensitive Protection

In addition to protecting data based on a selection of rows and / or columns, views can be used for **context-sensitive protection**. For example, you can create a view that gives a data entry clerk permission to access only those rows that he or she has added or updated. To do this, you would add a column to a table in which the user ID of the user entering each row is automatically recorded with a default. You can define this default in the **create table** statement as follows:

```
create table testtable
    (empid       int,
    startdate    datetime,
    username     varchar(30) default user)
```

Next, define a view that includes all the rows of the table where *uid* is the current user:

```
create view context_view
as
    select *
    from testtable
    where username = user_name()
with check option
```

The rows that are retrievable through this view depend on the identity of the person who issues the **select** command against the view. By adding the **with check option** to the view definition, you make it impossible for any data entry clerk to falsify the information in the *username* column.

Using Stored Procedures As Security Mechanisms

A user with permission to execute a stored procedure can do so, even if he or she does not have permissions on the tables or views referenced in that procedure, as long as the stored procedure and its referenced objects have the same owner. For example, a user might be given permission to execute a stored procedure that updates a row-and-column subset of a specified table, even though that user does not have any direct permissions on that table.

Index

Page numbers in **bold** are primary references.

E

F

G

H

I

J

L

M

S

T

U

V

W