Sybase SQL Server™
Security Administration Guide

Sybase SQL Server Release 11.0.x
Document ID: 35811-01-1100-02
Last Revised: December 15, 1995

Principal author: Barbara Grace

Document ID: 35811-01-1100

This publication pertains to Sybase SQL Server Release 11.0.x of the Sybase database management software and to any subsequent release until otherwise indicated in new editions or technical notes. 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.

Document Orders

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor.

Upgrades are provided only at regularly scheduled software release dates.

Sybase Trademarks

Restricted Rights

Table of Contents

5. Roles in SQL Server

6. Managing User Permissions

7. Managing Remote Servers

8. Auditing

List of Figures

List of Tables

Preface

This manual, the *Security Administration Guide*, describes how to administer security for Sybase SQL Server™ release 11.0. The manual explains how to use the security features provided by SQL Server to control user access to data.

Note that this manual is the primary component of the Trusted Facility Manual (TFM), designed to meet the Trusted Computer System Evaluation Criteria (TCSEC) requirements. Other components are:

- SQL Server installation and configuration guide
- *SQL Server System Administration Guide*

For information about these manuals, see "Related Documents" on page xvi.

Audience

This manual is intended for users who are administering SQL Server. The audience includes administrators who are concerned about adding users to the server, granting database access to users, establishing initial passwords, and administering auditing.

The manual assumes that readers have a general understanding of security and database administration but need to understand and administer the security features provided by SQL Server.

Note that this manual is not addressed to the general, non-administrative user with no special roles or privileges. The *Security Features User's Guide*, which is addressed to all users of SQL Server, provides instructions and guidelines for using the server in a secure manner.

How to Use This Book

This manual contains these chapters:

- Chapter 1, "Overview of Security Functions Provided by SQL Server," provides an overview of the security features that are available with SQL Server and directs you to the chapter where each feature is discussed in detail. The overview also introduces

the Trusted Computing Base (TCB) and its interface to users and client processes.

- Chapter 2, "Overall Process of Security Administration," provides an overall process for administering security and shows you where to read about each task.
- Chapter 3, "Getting Started After Installation," provides guidelines for setting up a secure operating environment after you have installed SQL Server.
- Chapter 4, "Managing SQL Server Logins and Database Users," describes methods for administering SQL Server login accounts and controlling user access to databases.
- Chapter 5, "Roles in SQL Server," lists the special roles supported by SQL Server and describes how to assign these roles to individual users.
- Chapter 6, "Managing User Permissions," describes the discretionary access control (DAC) functions provided by SQL 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 individual users.
- Chapter 7, "Managing Remote Servers," discusses the security implications of remote procedure calls (RPCs) and how to manage user access to procedures on remote servers.
- Chapter 8, "Auditing," explains how to record security-related system activity in an audit trail, how to use the audit trail to detect unauthorized access, and how to maintain the audit trail.

Related Documents

SQL Server relational database management system documentation is designed to satisfy both the inexperienced user's preference for simplicity and the experienced user's desire for convenience and comprehensiveness. The user's guide and the reference manuals address the various needs of end users, database and security administrators, application developers, and programmers.

Other manuals you may find useful are:

- SQL Server installation and configuration guide, which describes the installation procedures for SQL Server and documents operating system-specific system administration, security administration, and tuning tasks.

- *SQL Server Performance and Tuning Guide*, which explains how to tune SQL Server for maximum performance. The book includes information about database design issues that affect performance, query optimization, how to tune SQL Server for very large databases, disk and cache issues, and the effects of locking and cursors on performance.
- *SQL Server Reference Manual*, which contains detailed information on all of the commands and system procedures discussed in this manual.
- *SQL Server Reference Supplement*, which contains a list of the Transact-SQL® reserved words, definitions of system tables, a description of the *pubs2* sample database, a list of SQL Server error messages, and other reference information that is common to all the manuals.
- *SQL Server Security Features User's Guide*, which explains how to use the security features of SQL Server.
- *SQL Server System Administration Guide*, which provides in-depth information about administering servers and databases. The manual includes instructions and guidelines for managing physical resources and user and system databases, and specifying character conversion, international language, and sort order settings.
- SQL Server utility programs manual, which documents Sybase utility programs such as **isql** and **bcp**, which are executed at the operating system level.
- *Transact-SQL User's Guide*, which documents Transact-SQL, Sybase's enhanced version of the relational database language. It serves as a textbook for beginning users of the database management system.
- *What's New in Sybase SQL Server Release 11.0?*, which describes the new features in release 11.0.

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

The conventions for syntax statements in this manual are as follows:

Table 1: Syntax statement conventions

Key	Definition
command	Command names, command option names, utility names, utility flags, and other keywords are in **bold Helvetica** in paragraph text and **bold courier** in syntax statements.
variable	Variables, or words that stand for values that you fill in, are in *italics.*
{ }	Curly braces indicate that you choose at least one of the enclosed options. Do not include braces in your option.
[ ]	Brackets mean that choosing one or more of the enclosed options is optional. Do not include brackets in your option.
( )	Parentheses are to be typed as part of the command.
|	The vertical bar means you may select only one of the options shown.
,	The comma means you may choose as many of the options shown as you like, separating your choices with commas that are to be typed as part of the command.

- Syntax statements (displaying the syntax and all options for a command) are printed like this:

  ```
  sp_dropdevice [device_name]
  ```

  or, for a command with more options:

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

  In syntax statements, keywords (commands) are in non-italic font, and identifiers are in lowercase: non-italic font for keywords, 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)
  ```

- Case

You can disregard case when you type keywords:

```
SELECT is the same as Select is the same as select
```

Obligatory Options {You Must Choose At Least One}

- Curly braces and vertical bars: Choose **one and only one** option.

  ```
  {die_on_your_feet | live_on_your_knees |
  live_on_your_feet}
  ```

- Curly braces and commas: Choose one or more options. If you choose more than one, separate your choices with commas.

  ```
  {cash, check, credit}
  ```

Optional Options [You Don't Have to Choose Any]

- One item in square brackets: You don't have to choose it.

  ```
  [anchovies]
  ```

- Square brackets and vertical bars: Choose **none or only one**.

  ```
  [beans | rice | sweet_potatoes]
  ```

- Square brackets and commas: Choose **none, one, or more than one** option. If you choose more than one, separate your choices with commas.

  ```
  [extra_cheese, avocados, sour_cream]
  ```

Ellipsis: Do it Again (and Again)...

An ellipsis (. . .) means that you can **repeat** the last unit as many times as you like. In this syntax statement, **buy** is a required keyword:

```
buy thing = price [cash | check | credit]
    [, thing = price [cash | check | credit]]...
```

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 like. For each thing you buy, give its name, its price, and (optionally) a method of payment.

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
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

Help with your Sybase software is available in the form of documentation and Sybase Technical Support.

Each Sybase installation has a designated person who may contact Technical Support. If you cannot resolve your problem using the manuals, ask the designated person at your site to contact Sybase Technical Support.

1 Overview of Security Functions Provided by SQL Server

SQL Server is a database management system that is targeted for evaluation at the Class C2 criteria. 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."

The TCSEC defines a Trusted Computing Base (TCB) to be the collection of computer system protection mechanisms that are responsible for enforcing a security policy. SQL Server's TCB is a composite computing base that includes the operating system platform as a component. The SQL Server's TCB includes the underlying operating system, the server's executable code, internal data such as system databases and catalogs, Backup Server™, and **isql** and **bcp** (when these utilities are used as administrative interfaces).

Users interact with the TCB via:

- **isql**, the interactive command line interface for executing Transact-SQL commands.
- **bcp**, the bulk copy utility, which is used to import and export data into and out of database tables.
- System procedures, which are used to perform many system administration and security tasks.
- Communication through the Tabular Data Stream (TDS) protocol from a client process to SQL Server, and from SQL Server to the client process.

The Major Security Functions Available with SQL Server

SQL Server provides a number of security features that help you to protect sensitive data from inappropriate access and unauthorized disclosure. The major features are summarized in Table 1-1.

Table 1-1: Major security functions available with SQL Server

Security Feature	What It Is—In Short
Discretionary access controls (DAC)	Provides access controls that give object owners the ability to restrict access to objects, usually with the **grant** and **revoke** commands. This type of control is dependent upon an object owner's discretion.
Identification and authentication controls	Ensures that only authorized users can log into the system.
Division of roles	Allows you to grant privileged roles to specified users so that only designated users can perform critical management tasks.
Auditing	Provides the capability to audit logins, logouts, server boot operations, remote procedure calls, role changes, privileged commands, system errors, access to databases, tables, views, and stored procedures, and all actions for a specific user.

The following sections provide an overview of each major function and refer you to the chapter where the function is covered in detail.

Discretionary Access Controls

The SQL commands **grant** and **revoke** provide **discretionary** protection to SQL commands and objects. Owners of objects can, at their discretion, grant access to the objects to other users. The object owners can also grant other users the ability to pass the access permission to other users. With SQL Server's discretionary access controls, you can give various kinds of permissions to users, groups, and roles with the **grant** command. The **revoke** command permits you to rescind these permissions. The **grant** and **revoke** commands give users permission to execute specified commands 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 (as System Administrator, Database Owner, or database object owner), and by whether or not a particular user has been granted a permission with the option to grant that permission to other users.

Discretionary access controls are discussed fully in Chapter 6, "Managing User Permissions."

Identification and Authentication Controls

Every user in SQL 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.

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

Identification and authentication controls are discussed in Chapter 4, "Managing SQL Server Logins and Database Users" and, for remote servers, in Chapter 7, "Managing Remote Servers."

Division of Roles

An important feature in SQL Server is the division of **roles**. The roles supported by SQL Server enable you to enforce and maintain individual accountability. Various security-related, administrative, and operational tasks are grouped into the following roles:

- System Security Officer, who performs security-related tasks such as:
  - Creating server login accounts, which includes assigning initial passwords
  - Changing the password of any account
  - Granting and revoking the System Security Officer and Operator roles
  - Setting the password expiration interval
  - Managing the audit system
- System Administrator, whose tasks include:
  - Managing disk storage
  - Monitoring SQL Server's automatic recovery procedure
  - Fine-tuning SQL Server by changing the configurable system parameters

  - Diagnosing system problems and reporting them as appropriate
  - Backing up and loading databases
  - Granting and revoking the System Administrator role
  - Modifying and dropping server login accounts
  - Granting permissions to SQL Server users
  - Creating user databases and granting ownership of them
  - Setting up groups (which are convenient in granting and revoking permissions)
- Operator, 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 one.

These roles provide individual accountability for users performing administrative tasks. Their actions can be audited and attributed to them. Division of roles is discussed fully in Chapter 5, "Roles in SQL Server."

Auditing

A comprehensive audit system is provided with SQL Server. The audit system consists of a system database called *sybsecurity* and a set of system procedures that allow you to selectively set the audit options you need.

You can choose to audit the following:

- Within a server you can audit logins and logouts, server boots, RPC connections made from other servers, system errors, and execution of commands requiring special roles.
- At the database level, you can audit the use of the **grant**, **revoke**, **truncate table**, and **drop** commands within a database; the use of the **drop** and **use** commands on a database; and references to a specific database from within another database.
- At the user level, you can audit a specified user's attempts to access tables and views, and you can audit the text of commands submitted to the server by a user.
- At the object level, you can audit access to specified tables and views and execution of stored procedures and triggers.

- If you have the correct permissions, you can send ad hoc annotations to the audit trail.

Auditing functions are discussed fully in Chapter 8, “Auditing.”

2 Overall Process of Security Administration

This chapter presents an overall process for administering SQL Server in a secure manner. Table 2-1 lists the major tasks that are required and refers you to detailed instructions for handling each task.

Table 2-1: Overall process for security administration

Task	Description	Refer To
Install SQL Server, including auditing	This task includes the procedures that you follow to install SQL Server. The task includes preparing for installation, loading files from your distribution medium, performing the actual installation, and getting started administering the physical resources that are required.	SQL Server installation and configuration guide.
Get started with security administration	Immediately after installation, take steps to set up a secure administrative environment for SQL Server. This includes enabling auditing, granting roles to individual users to ensure individual accountability, and assigning login names to System Administrators and System Security Officers.	Chapter 3, "Getting Started After Installation," in this book.
Determine the physical resources required for your user databases and create the databases	This task includes making storage management decisions, initializing database devices, creating and using segments, and creating user databases.	*System Administration Guide.*
Add user logins to the server and add users to databases	Involves adding logins, creating groups, add ing users to databases, dropping and locking logins, and assigning initial passwords.	Chapter 4, "Managing SQL Server Logins and Database Users," in this book.
Establish and assign special roles to users	Includes assigning these roles: **sa_role** (for System Administrators), **sso_role** (for System Security Officers), and **oper_role** (for operators who backup and restore databases).	Chapter 5, "Roles in SQL Server," in this book.
Administer permissions for users, groups, and roles	Includes granting and revoking permissions to execute certain SQL commands, execute certain system procedures, and access databases, tables, particular table columns, and views.	Chapter 6, "Managing User Permissions," in this book.

Table 2-1: Overall process for security administration (continued)

Task	Description	Refer To
Administer the use of remote servers	Includes establishing and administering the access that is permitted between servers, adding and dropping remote server access, and mapping remote login names to local login names.	Chapter 7, "Managing Remote Servers," in this book, coupled with the SQL Server installation and configuration guide.
Set up and maintain auditing	Includes determining what is to be audited, auditing the use of SQL Server, and using the audit trail to detect penetration of the system and misuse of resources.	Chapter 8, "Auditing," in this book.

3 Getting Started After Installation

Status of SQL Server After Installation

This chapter assumes that you have already installed SQL Server. It provides guidelines for what to do following installation.

When SQL Server is installed, a single login called "sa", with a NULL password, is automatically configured with the System Administrator and System Security Officer roles.

➤ ***Note***

The SQL Server installation and configuration guide recommends that you change the password of the "sa" login, so the password might no longer be NULL. If the password is still NULL, use **sp_password** to change it as soon as possible after installation. For example, to change the password to "16tons", execute:

```
sp_password null, "16tons"
```

Getting Started Administering SQL Server

The "sa" account is configured with the System Administrator and System Security Officer roles, so the account, essentially, has unlimited power. To limit the power of any one login account and to establish individual accountability, use the "sa" account initially to assign the System Security Officer and System Administrator roles to individual users. Then lock the "sa" account so that no user has unlimited access. Consider following the general guidelines listed in the next section to help ensure that your operating environment is secure. Specific steps you can take and an overall example follow the list of guidelines.

Guidelines

Follow these guidelines when you start administering SQL Server:

- Use the "sa" login only during initial setup. Instead of allowing several users to use the "sa" account, establish individual accountability by assigning specific roles to individual administrators.

- Enable auditing early in the administration process so that you have a record of privileged commands that are executed by System Security Officers and System Administrators. You might also want to audit commands that are executed by those with other special roles, such as Operators when they dump and load databases.
- Assign SQL Server login names that are the same as their respective operating system login names. This makes login to SQL Server easier, simplifies management of server and operating system login accounts, and makes it easier to correlate the audit data generated by SQL Server with that of the operating system.

Steps to Take

To follow these guidelines, take the steps listed in Table 3-1 after installation. Be sure to read the sections of this manual that give detailed instructions for each step. For example, before setting up auditing, read Chapter 8, "Auditing," and before assigning roles, read Chapter 5, "Roles in SQL Server." For an example of how to use the "sa" account to set up a secure operating environment, see "An Example of Getting Started" on page 3-4.

➤ ***Note***

If you have decided not to install auditing, skip the steps that pertain to auditing.

Table 3-1: Steps to get started

Step	Commands or Utilities to Use	Refer To
Log into your operating system.	This is specific to your operating system.	Your operating system documentation.
Log into SQL Server as "sa". In SQL Server, both the **sa_role** and **sso_role** are active when you log in as "sa".	**isql utility** Example: **isql -Usa**	SQL Server utility programs manual.

Table 3-1: Steps to get started (continued)

Step	Commands or Utilities to Use	Refer To
Enable auditing for privileged commands. **Note:** Before you can do this step, auditing must be installed. See the SQL Server installation and configuration guide for information about how to install auditing.	**sp_auditoption** Example: **sp_auditoption "enable auditing", "on"** **sp_auditoption "sa commands", "both"** Required: To execute **sp_auditoption**, you must have the **sso_role** active.	Chapter 8, "Auditing."
Add logins and passwords for System Security Officers, System Administrators, and Operators. Make every SQL Server user's login name the same as the user's operating system login name.	**sp_addlogin** Example: **sp_addlogin ralph, orientab, @fullname ="Ralph Smith"** Required: To execute **sp_addlogin**, you must have the **sso_role** active.	Chapter 4, "Managing SQL Server Logins and Database Users."
Grant the **sa_role**, **sso_role**, and **oper_role** to System Administrators, System Security Officers, and Operators, respectively.	**sp_role** Example: **sp_role "grant","sso_role", ralph** Required: To grant the **sso_role** or **oper_role**, you must have the **sso_role** active. To grant the **sa_role**, you must have the **sa_role** active.	Chapter 5, "Roles in SQL Server."
Give access to the *sybsecurity* database to the System Security Officer or Officers who are responsible for administering auditing.	If you have a single auditing administrator, use **sp_changedbowner** Example: **use sybsecurity** **sp_changedbowner ralph** If you have more than one auditing administrator, use **sp_adduser** and **grant** to give permissions for *sybsecurity* to the other auditing administrators.	For changing database ownership, see **sp_changedbowner** in the *SQL Server Reference Manual*. To add users to databases, see Chapter 4, "Managing SQL Server Logins and Database Users." To grant permissions, see Chapter 6, "Managing User Permissions."

Table 3-1: Steps to get started (continued)

Step	Commands or Utilities to Use	Refer To
Verify that you can log into SQL Server as the users you have configured with the **sa_role** and **sso_role**. Then, lock the "sa" account.	**sp_locklogin** Example: **sp_locklogin sa,"lock"** Required: You must have either the **sa_role** or the **sso_role** active.	Chapter 4, "Managing SQL Server Logins and Database Users."
Be sure to check all scripts that may contain the "sa" login name and password. These can include scripts that perform backups, run **bcp**, or perform **dbcc** checking. The scripts cannot run if they are meant to run as "sa" and that account is locked. Change the logins in those scripts to the name of a user with the correct role.	Use your favorite text editor for this step.	

➤ ***Note***

To use the "sa" account later for a SQL Server upgrade, you must unlock the account. After completing the upgrades, you can again lock the account. For information about how to lock and unlock accounts, see "Locking SQL Server Logins: sp_locklogin" on page 4-12. See the SQL Server installation and configuration guide for information about upgrading SQL Server.

An Example of Getting Started

Suppose you have decided to assign special roles to these users:

Table 3-2: Example of getting started

Name	Role	Operating System Login Name
Ralph Smith	**sso_role**	ralph
Kathy Masters	**sa_role**	kathy
Mary Randolph	**sa_role**	mary
Gene Wishing	**oper_role**	gene

The following sequence of commands provides an example of using the initial "sa" account to set up a secure operating environment for

SQL Server, based upon the role assignments shown in Table 3-2 on page 3-4.

Table 3-3: Commands to set up a secure operating environment

isql Utility and SQL Commands	What the Commands Are Doing
isql -Usa	Logs into SQL Server as "sa". Both **sa_role** and **sso_role** are active.
sp_auditoption "enable auditing","on" **sp_auditoption "sa commands","both"** **sp_auditoption "sso commands","both"** **sp_auditoption "oper commands","both"**	Enables auditing for privileged commands. All commands that require **sa_role**, **sso_role**, or **oper_role** are audited. If the **sso_role** is active, you can use the **select** command to query the *sysaudits* table, which contains the audit log. To access *sysaudits*, you must be in the *sybsecurity* database.
sp_addlogin ralph, orientab, @fullname ="Ralph Smith" **sp_addlogin kathy, grace33, @fullname ="Kathy Masters"** **sp_addlogin mary, guten33, @fullname ="Mary Randolph"** **sp_addlogin gene, sleeping, @fullname ="Gene Wishing"**	Adds logins and passwords for Ralph, Kathy, Mary, and Gene. A default database is not specified for any of these users, so their default database is *master*.
sp_role "grant","sso_role", ralph **sp_role "grant","sa_role", mary** **sp_role "grant","sa_role", kathy** **sp_role "grant","oper_role", gene**	Grants the **sso_role** to Ralph, the **sa_role** to Mary and Kathy, and the **oper_role** to Gene.
use sybsecurity **sp_changedbowner ralph**	Grants access to the auditing database, *sybsecurity*, by making Ralph, who is the System Security Officer, the database owner.
sp_locklogin sa,"lock"	Locks the "sa" login. Now, no one can login as "sa". Individuals can assume only the roles that are configured for them.

➤ ***Note***

Do not lock the "sa" account until you have configured individual users with the **sa_role** and **sso_role** and have verified that the roles operate successfully.

4 Managing SQL Server Logins and Database Users

Introduction

This chapter describes methods for managing SQL Server login accounts and database users. Topics covered include:

- Adding new logins to SQL Server
- Creating groups
- Adding users to databases
- Dropping logins, users, and groups
- Locking SQL Server logins
- Changing user information
- Changing user information such as passwords and group membership
- Obtaining information about users, groups, and SQL Server usage

Adding New Users: an Overview

The process of adding new logins to SQL Server, adding users to databases, and granting them **permission** to use commands and database objects is divided among the System Security Officer, System Administrator, and Database Owner.

The process of adding new users consists of the following steps:

1. A System Security Officer creates a server login account for a new user with **sp_addlogin.**
2. A System Administrator or Database Owner adds a user to a database with **sp_adduser.** This command can also give the user an alias or assign the user to a group. Groups are added using **sp_addgroup**. See “Creating Groups: sp_addgroup” on page 4-5 for more information.
3. A System Administrator, Database Owner, or object owner grants the user or group specific permissions on specific commands and database objects. Users or groups can also be granted permission to grant specific permissions on objects to other users or groups. See Chapter 6, “Managing User Permissions” for detailed information about permissions.

The following table summarizes the commands and system procedures used for these tasks.

Table 4-1: System procedures for adding users to SQL Server and databases

Command or Procedure	Task	Executed By	Where
sp_addlogin	Create new logins, assign passwords, default databases, default language, and full name	SSO	Any database
sp_addgroup	Create groups	DBO or SA	User database
sp_adduser	Add users to database, assign aliases, and assign groups	DBO or SA	User database
grant	Grant groups or users permission to create or access database objects	DBO, SA, or object owner	User database

Choosing a Password

Your password is the first line of defense against SQL Server access by unauthorized people. SQL Server passwords must be at least six bytes long and can contain any printable letters, numerals, or symbols. When you create your own password or one for another user, choose one that cannot be guessed. Do not use personal information, names or nicknames of pets or loved ones, or words that appear in the dictionary. The most secure passwords are ones that combine uppercase and lowercase characters with numbers, punctuation, and accented characters. Once you select a password, protecting it is your responsibility. Never give anyone your password and never write it down where anyone can see it.

Adding Logins to SQL Server

The system procedure **sp_addlogin** adds new **login** names to SQL Server. It does not give the user permission to access any user databases; the procedure **sp_adduser** gives users access to specific databases. Only the System Security Officer can execute **sp_addlogin**. Here is its syntax:

```
sp_addlogin loginame, passwd [, defdb]
   [, deflanguage [, fullname]]]
```

- The *loginame* parameter (the new user's login name) is required. The login name must follow the rules for identifiers and must be unique on SQL Server. To simplify both the login process and server administration, make the SQL Server login name the same as the user's operating system login name. This makes logging into SQL Server easier, since many client programs use the operating system login name as a default. It also simplifies management of server and operating system login accounts, and makes it easier to correlate usage and audit data generated by SQL Server and by the operating system.
- The *passwd* parameter, also required, is a password for the new user. The password must be at least six bytes long, and can be any printable letters, numerals, or symbols. A password must be enclosed in quotation marks in **sp_addlogin** if:
  - It includes characters besides A-Z, a-z, 0-9,_, #, valid single- or multi-byte alphabetic characters, or accented alphabetic characters
  - It begins with 0-9

  See "Choosing a Password" on page 4-2 for guidelines on choosing secure passwords. A user can change his or her own password, and a System Security Officer can change any user's password, with **sp_password**.
- *defdb* specifies the name of the new user's **default database**. This causes the user to start each session on SQL Server in that database, without having to issue the **use** command. A System Administrator can change anyone's default database with **sp_modifylogin**; other users can only change their own.

➤ ***Note***

If you do not specify a user's default database parameter with **sp_addlogin**, the user's default database is *master*. Assign default databases other than *master* to most users, to discourage users from creating database objects in the *master* database.

After you specify the default database, add the user to the default database with **sp_adduser**, so that he or she can log in directly to the default database.

- *deflanguage* specifies the **default language** in which this user's prompts and messages are displayed. If you omit this parameter, SQL Server's default language is used. The System Administrator

can change any user's default language with **sp_modifylogin**; other users can only change their own.

- *fullname* specifies a full name for the user. This can be useful for documentation and identification purposes. If omitted, no full name is added. The System Administrator can change any user's full name with **sp_modifylogin**; other users can only change their own.

The following statement sets up an account for the user "maryd" with the password "100cents," the default database (*master*), the default language, and no full name:

```
sp_addlogin "maryd", "100cents"
```

Quotation marks are not required around character-type parameters to stored procedures as long as they do not contain spaces, punctuation marks, or Transact-SQL reserved words, or do not begin with a digit (0–9).

Once this statement is executed, "maryd" can log in to the SQL Server. She is automatically treated as a guest user in *master*, with limited permissions, unless she has been specifically given access to *master*.

Here is a statement that sets up a login account and a password ("rubaiyat") for user "omar_khayyam," and makes *pubs2* his default database:

```
sp_addlogin omar_khayyam, rubaiyat, pubs2
```

If you wish to specify a full name for a user but use the default database and language, you must specify **null** in place of those two parameters. For example:

```
sp_addlogin omar, rubaiyat, null, null,
    "Omar Khayyam"
```

As an alternative, you can specify a parameter name, in which case you do not have to specify all the parameters. For example:

```
sp_addlogin omar, rubaiyat,
    @fullname = "Omar Khayyam"
```

Effects of *sp_addlogin* on System Tables

When you execute **sp_addlogin**, SQL Server adds a row to *master.dbo.syslogins*, assigns a unique server **user ID** (*suid*) for the new user, and fills in other information. When a user logs in, SQL Server looks in *syslogins* for the name and password the user gives.

The *password* column is encrypted with a one-way algorithm and so is not human-readable.

The *suid* column in *syslogins* uniquely identifies the user on SQL Server. A user's *suid* remains the same no matter what database he or she is using. The *suid* 1 is always assigned to the default "sa" account that is created when SQL Server is installed. Other users' server user IDs are small integers assigned consecutively by SQL Server each time **sp_addlogin** is executed.

Permissions Required

Only a System Security Officer can execute **sp_addlogin**.

Creating Groups: *sp_addgroup*

Groups provide a convenient way to grant and revoke permissions to more than one user in a single statement. Groups enable you to provide a collective name to a group of users. They are especially useful if you administer a SQL Server installation that has large numbers of users.

Every user is a member of the group "public," and can be a member of one other group. (Users remain in "public" whether or not they belong to another group.)

It is probably most convenient to create groups before adding users to a database since the **sp_adduser** procedure can assign users to groups as well as add them to the database. A System Administrator or the Database Owner can create a group at any time with **sp_addgroup**. The syntax for **sp_addgroup** is:

```
sp_addgroup grpname
```

The group name, a required parameter, must follow the rules for identifiers. For example, to set up the Senior Engineering group, use the following command while using the database to which you wish to add the group:

```
sp_addgroup senioreng
```

The System Administrator can assign or re-assign users to groups with **sp_changegroup**.

Permissions Required

sp_addgroup can be executed by the Database Owner or by a System Administrator.

Effects of *sp_addgroup* on System Tables

The **sp_addgroup** system procedure adds a row to *sysusers* in the current database. In other words, each group in a database—as well as each user—has an entry in *sysusers*.

Adding Users to Databases: *sp_adduser*

The procedure **sp_adduser** adds a user to a specific database. The user must already have a SQL Server login. Here is the syntax for **sp_adduser**:

```
sp_adduser loginame [, name_in_db [, grpname]]
```

- *loginame* is the login name of an existing user. It is required.
- *name_in_db* allows you to specify a name different from the login name by which the user is to be known inside the database. It is optional.

  This feature is supplied to accommodate users' preferences. For example, if there are five SQL Server users named Mary, all of them must have different login names. Mary Doe might log in as "maryd", Mary Jones as "maryj", and so on. However, if the Marys don't use the same databases, each might prefer to be known simply as "mary" inside a particular database.

  If no *name_in_db* parameter is given, the name inside the database is the same as *loginame*.

  Be careful not to confuse this capability with the alias mechanism described in "Using Aliases in Databases" on page 4-17, which maps the identity and permissions of one user to another.
- *grpname*, also optional, is the name of an existing group in the database. If you do not specify a group name, the user is made a member of the default group "public". Users remain in "public" even if they are a member of another group. See "Changing a User's Group Membership: sp_changegroup" on page 4-16 for information about modifying a user's group membership.

Here is a command that the owner of the *pubs2* database could give to allow "maryh" of the (already existing) engineering group to access *pubs2*:

```
sp_adduser maryh, mary, eng
```

Here is how to give "maryd" access to *pubs2*, with her name in the database the same as her login name, and making her a member of the "public" group:

```
sp_adduser maryd
```

Here is how to add a user and put the user in a group, but without different names, using **null** in place of a new user name:

```
sp_adduser maryj, null, eng
```

Users who have access to a database still need permission to do things inside it: to read data, modify data, and use certain commands. These permissions are set up by the owners of the database objects or user databases with the **grant** and **revoke** commands. They are discussed in Chapter 6, "Managing User Permissions".

Effects of *sp_adduser* on System Tables

The **sp_adduser** system procedure adds a row to the *sysusers* system table in the current database. Once a user has an entry in the *sysusers* table of a database, he or she:

- Can issue the **use** command and access that database
- Will use that database by default, if the default database parameter was issued as part of **sp_addlogin**
- Can use **sp_modifylogin** to make that database the default

Permissions Required

sp_adduser can be executed by the Database Owner and System Administrator.

Adding a "guest" User

Creating a user named "guest" in a database enables any user with a SQL Server account to access the database as a **guest** user. If a user issues the **use *database_name*** command, and his or her name is not found in the database's *sysusers* or *sysalternates* table, SQL Server

looks for a guest user. If there is one, the user is allowed to access the database, with the permissions of the guest user.

The Database Owner can add a guest entry to the *sysusers* table of the database with the system procedure **sp_adduser**:

```
sp_adduser guest
```

The guest user can be removed with **sp_dropuser**, as discussed in "Dropping Database Users: sp_dropuser" on page 4-11.

If you drop the guest user from the *master* database, server users who have not yet been added to any databases will be unable to log into SQL Server.

➤ ***Note***

Even though more than one individual can be a guest user in a database, you can still maintain individual accountability by auditing the database operations that are performed by a guest user. SQL Server maintains individual accountability by using the user's server user ID, which is unique within the server, to audit each user's activity. This ID is stored in the *suid* column of the *master..syslogins* table. For more information about auditing, see Chapter 8, "Auditing."

"guest" User Permissions

When you first add the user name "guest" to a database, "guest" inherits the privileges of "public". The Database Owner and the owners of database objects can change these permissions as they wish, using **grant** and **revoke**, to make the privileges of "guest" either more restrictive or less restrictive than those of "public." See Chapter 6, "Managing User Permissions," for a description of the privileges "public" has.

When you install SQL Server, *master..sysusers* contains a guest entry. The installation script for the *pubs2* database also contains a guest entry for its *sysusers* table.

"guest" User in User Databases

In user databases, the Database Owner is responsible for setting up any guest mechanisms that are needed. Adding a guest user to a user database allows an owner to permit all SQL Server users to use that

database without having to use **sp_adduser** to explicitly name each one as a database user.

You can use the guest mechanism to restrict access to database objects while allowing access to the database.

For example, the owner of the *titles* table could grant **select** permission on *titles* to all database users except "guest" by executing the following three commands:

```
grant select on titles to public
sp_adduser guest
revoke all on titles from guest
```

If you wish to grant permissions to "public", but do not wish to grant these permissions to "guest", be sure to issue a **revoke** command after granting permission to "public". The group "public" includes all users.

"guest" User in *pubs2*

The "guest" user entry in *pubs2*, the sample database, allows new SQL Server users to follow the examples in the *Transact-SQL User's Guide* and to engage in a certain amount of experimentation. The guest in *pubs2* is given a wide range of privileges:

- **select** permission and data modification permission on all of the user tables
- **execute** permission on all of the procedures
- **create table, create view, create rule, create default** and **create procedure** permission

Visitor Accounts on SQL Server

As another method of accommodating visiting users on SQL Server, the System Security Officer can use **sp_addlogin** to enter a row in *master..syslogins* with a login name and password that visiting users are instructed to use (for example, "visitor"). Typically, such users are granted very restricted permissions. A default database may be assigned.

◆ ***WARNING!***

Setting up a visitor account to be used by more than one user is not recommended because you lose individual accountability. All users of the visitor account have the same server user ID; therefore, you cannot audit individual activity. Be careful not to confuse such visitor accounts with the guest user mechanism described earlier. Because each guest user has a unique server ID, you can audit individual activity and maintain individual accountability.

Adding Remote Users

You can allow users on another SQL Server to execute stored procedures on your server by enabling remote access. Working with a System Administrator of the remote server, you can also allow users of your server to execute **remote procedure calls** to the remote server. To enable remote procedure calls, both the local and the remote server must be configured, following several steps. See Chapter 7, "Managing Remote Servers," for information about setting up remote servers and adding remote users.

Assigning Roles and Granting Permissions to Database Users

The final steps in adding database users are assigning them special roles, as required, and granting them permission to use commands and database objects. See Chapter 5, "Roles in SQL Server," and Chapter 6, "Managing User Permissions."

Dropping Logins, Users, and Groups

The following system procedures allow a System Administrator or Database Owner to drop logins, users and groups.

Table 4-2: Dropping logins, users, and groups

System Procedure	Task	Executed By	Where
sp_droplogin	Drop user from SQL Server	SA	*master*
sp_dropuser	Drop user from database	DBO or SA	User database

Table 4-2: Dropping logins, users, and groups (continued)

System Procedure	Task	Executed By	Where
sp_dropgroup	Drop group from database	DBO or SA	User database

Dropping Logins: *sp_droplogin*

The system procedure **sp_droplogin** denies a user access to SQL Server. It may be easier to lock logins rather than drop them, for these reasons:

- You cannot drop a login who is a user in any database, and you cannot drop a user from a database if the user owns any objects in that database or has granted any permissions on objects to other users.
- SQL Server may reuse the dropped login account's server user ID (*suid*) when the next login account is created. This only occurs when the dropped login holds the highest *suid* in *syslogins*, but could compromise accountability if execution of **sp_droplogin** is not being audited.
- You cannot drop the last remaining System Security Officer's or System Administrator's login account.

See "Locking SQL Server Logins: sp_locklogin" on page 4-12 for information about locking logins.

Here is the syntax for **sp_droplogin**:

```
sp_droplogin loginame
```

Only the System Administrator can execute **sp_droplogin.**

Dropping Database Users: *sp_dropuser*

The system procedure **sp_dropuser** denies a SQL Server user access to the database in which **sp_dropuser** is executed. (If there is a "guest" user defined in that database, the user can still access that database as "guest".)

Here is the syntax for dropping a user from a database:

```
sp_dropuser name_in_db
```

name_in_db is usually the login name, unless another name has been assigned.

sp_dropuser can be executed only by the Database Owner or a System Administrator.

➤ ***Note***

You cannot drop a user who owns objects. Since there is no command to transfer ownership of objects, you must drop objects owned by a user before you drop the user with **sp_dropuser**. If you wish to deny access to a user who owns objects, use **sp_locklogin** to lock his or her account.

You also cannot drop a user who has granted permissions to other users. **sp_dropuser** displays an informational message when you attempt to do this. Use **revoke with cascade** to revoke permissions from all users who were granted permissions by the user to be dropped, then drop the user. You must then re-grant permissions to the users, if appropriate.

Dropping Groups: *sp_dropgroup*

To drop a group, use **sp_dropgroup** followed by the group name:

```
sp_dropgroup grpname
```

You cannot drop a group that has members. If you try to do so, the error report displays a list of the members of the group you are attempting to drop. To remove users from a group, execute **sp_changegroup**, discussed in "Changing a User's Group Membership: sp_changegroup" on page 4-16.

Locking SQL Server Logins: *sp_locklogin*

Locking a SQL Server login account prevents that user from logging in. You may prefer to lock accounts instead of dropping them for reasons discussed in "Dropping Logins: sp_droplogin" on page 4-11. The **sp_locklogin** system procedure locks and unlocks accounts, or displays a list of locked accounts.

The syntax is:

```
sp_locklogin [loginame, "{lock | unlock}"]
```

- *loginame* is the name of the account to be locked or unlocked. It must be an existing, valid account.
- **lock** | **unlock** specifies whether the account is to be locked or unlocked.

Using **sp_locklogin** with no parameters displays a list of all locked logins.

You can lock an account that is currently logged in. When you do this, SQL Server displays a warning that the user is not locked out of the account until he or she logs out. You can lock the account of a Database Owner, and a locked account can own objects in databases. In addition, you can use **sp_changedbowner** to specify a locked account as the owner of a database.

SQL Server ensures that there is always at least one unlocked System Security Officer's account and one unlocked System Administrator's account. When you attempt to lock the last unlocked account of a System Administrator or a System Security Officer, SQL Server displays an error message and does not lock the account.

Permissions Required

You must be a System Administrator or a System Security Officer to use **sp_locklogin** to lock or unlock accounts.

Changing User Information

Additional system procedures allow you to change any of the user information added with the commands discussed earlier in this chapter. For example, **sp_modifylogin** and **sp_password** change default databases and passwords for SQL Server users.

The system procedures described in Table 4-3 change aspects of users' logins and database usage.

Table 4-3: System procedures for changing user information

System Procedure	Task	Executed By	Where
sp_password	Change another user's password	SSO	Any database
	Change own password	user	
sp_modifylogin	Change a login account's default database, default language, or full name	SA	Any database
	Change your own default database, default language, or full name	user	
sp_changegroup	Change group assignment of a user	SA, DBO	User database

Changing Passwords: *sp_password*

Your site may choose to use the **systemwide password expiration** configuration parameter to establish a password expiration interval to force all SQL Server users to change their passwords on a regular basis. For information about how to set the configuration parameter, see Chapter 11, "Setting Configuration Parameters," in the *System Administration Guide*. Even if you do not use **systemwide password expiration**, for security reasons it is important that users change their passwords from time to time.

The column *pwdate* in *syslogins* records the date of the last password change. This query selects all login names whose passwords have not been changed since January 30th, 1994:

```
select name, pwdate
from syslogins
where pwdate < "Jan 30 1994"
```

A user can change passwords at any time with the system procedure **sp_password.** The System Security Officer can use this system procedure to change any other user's password. The syntax is:

sp_password ***caller_passwd, new_passwd*** **[,** ***loginame*****]**

- *caller_passwd* is the password of the login account that is currently executing **sp_password**. When you are changing your own password, this is your old password. When a System Security Officer uses **sp_password** to change another user's password, *caller_passwd* is the password of the System Security Officer.
- *new_passwd* is the new password for the user executing **sp_password**, or for the user indicated by *loginame*. The password must be at least six bytes long, and can be any printable letters, numerals, or symbols. A password must be enclosed in quotation marks if:
  - It includes characters other than A-Z, a-z, 0-9, _, #, valid single- or multi-byte alphabetic characters, or accented alphabetic characters
  - It begins with 0-9

  See "Choosing a Password" on page 4-2 for guidelines on selecting a password.
- *loginame* may only be used by a System Security Officer to change another user's password.

For example, a user can change her password from "3blindmice" to "2mediumhot" with this command:

```
sp_password "3blindmice", "2mediumhot"
```

These passwords are enclosed in quotes because they begin with numerals.

In the following example, the System Security Officer whose password is "2tomato" changes Victoria's password to "sesame1":

```
sp_password "2tomato", sesame1, victoria
```

Null Passwords

You may not assign a null password. However, when SQL Server is installed, the default "sa" account has a null password. The following example changes a null password to a valid one:

```
sp_password null, "16tons"
```

Note that "null" is not enclosed in quotes.

Permissions Required

All users can use **sp_password** to change their own passwords. Only a System Security Officer can use it to change another user's password.

Changing User Defaults with *sp_modifylogin*

The **sp_modifylogin** system procedure changes a user's default database, default language, or full name. Any user can use **sp_modifylogin** to change his or her own login, and a System Administrator can use **sp_modifylogin** to change these settings for any user. The syntax is:

sp_modifylogin ***account, column, value***

- *account* is the name of the user whose account you are modifying.
- *column* specifies the option that you are changing. The options are listed in Table 4-4.

Table 4-4: Options for sp_modifylogin

Option	Definition
defdb	The "home" database to which the user is connected when he or she logs in.
deflanguage	The official name of the user's default language, as stored in *master..syslanguages*.
fullname	The user's full name.

- *value* is the new value for the specified option.

After you execute **sp_modifylogin** to change the default database, the user is connected to the new default database the next time he or she logs in. However, **sp_modifylogin** does not automatically give the user access to the database. Unless the Database Owner has set up access with **sp_adduser**, **sp_addalias**, or with a guest user mechanism, the user is connected to *master* even after his or her default database has been changed.

Examples

This example changes the default database for "anna" to *pubs2*:

```
sp_modifylogin anna, defdb, pubs2
```

This example changes the default language for "claire" to French:

```
sp_modifylogin claire, deflanguage, french
```

This example changes the full name for "clemens" to "Samuel Clemens."

```
sp_modifylogin clemens, fullname, "Samuel Clemens"
```

Changing a User's Group Membership: *sp_changegroup*

The System Administrator or the Database Owner can use the system procedure **sp_changegroup** to change an existing user's group affiliation. At any one time, each user can be a member of only one group other than "public," of which all users are always members. When new users are added to the database, they are assigned to the group "public" as well as any group specified with the **sp_adduser** procedure.

Before you execute **sp_changegroup**:

- The group must exist. (Use **sp_addgroup** to create a group.)

- The user must have access to the current database (must be listed in *sysusers*).

The syntax for **sp_changegroup** is:

sp_changegroup ***grpname, username***

For example, to change user "jim" from his current group to the group "manage", execute:

```
sp_changegroup manage, jim
```

To remove a user from a group without assigning the user to another group, you must change the group affiliation to "public":

```
sp_changegroup "public", jim
```

The name "public" must be in quotes because it is a reserved word. All users are always members of "public". This command reduces Jim's group affiliation to "public" only.

When a user changes from one group to another, the user loses all permissions that he or she had as a result of belonging to the old group, but gains the permissions that have been granted to the new group.

The assignment of users into groups can be changed at any time.

Using Aliases in Databases

The alias mechanism allows you to treat two or more users as the same user inside a database, so that they all have the same privileges. It is often used so that more than one user can assume the role of Database Owner. Be aware, however, that a Database Owner has a large number of permissions within a database. These permissions include the capability to use **setuser** to impersonate another user in the database. The alias mechanism can also be used to set up a collective user identity, within which the identities of individual users can be traced by auditing their activities.

➤ ***Note***

Even though more than one individual can use the alias in a database, you can still maintain individual accountability by auditing the database operations performed by each user. SQL Server maintains individual accountability by using the user's server user ID, which is unique within the server, to audit each user's activity. This ID is stored in the *suid* column of the *master..syslogins* table. For more information about auditing, see Chapter 8, "Auditing."

For example, suppose that several vice presidents want to use a database with identical privileges and ownerships. One way to accomplish this is to add the login "vp" to SQL Server and the database; each vice president logs in as "vp". The problem with this method is that there is no way to tell the individual users apart. The other approach is to alias all the vice presidents, each of whom has his or her own SQL Server account, to the database user name "vp".

The following system procedures are used to manage aliases:

Table 4-5: System procedures for managing aliases

System Procedure	Task	Executed By	Where
sp_addalias	Add an alias for a user	DBO or SA	User database
sp_dropalias	Drop an alias	DBO or SA	User database

Adding Aliases: *sp_addalias*

The syntax for **sp_addalias** is:

sp_addalias *loginame, name_in_db*

- *loginame* is the name of the user who wants an alias in the current database. This user must have an account on SQL Server but cannot be a user in the current database.
- *name_in_db* is the name of the database user to whom the first user wishes to be linked. This name must exist in both *master..syslogins* and in *sysusers* in the current database.

Both parameters are required.

Executing **sp_addalias** maps the user with the specified login name to the user with the specified *name_in_db*. It does this by adding a row to the system table *sysalternates*.

When a user tries to **use** a database, SQL Server checks for the user's server user ID number (*suid*) in *sysusers*. If it is not found, SQL Server then checks *sysalternates*. If the user's *suid* is found there, mapped to a database user's *suid*, the first user is treated as the second user while using the database.

As an example, suppose that Mary owns the database. She wishes to allow both Jane and Sarah to use the database as if they were its owner. Jane and Sarah have logins on SQL Server but are not authorized to use Mary's database. Mary executes these commands:

```
sp_addalias jane, dbo
exec sp_addalias sarah, dbo
```

Now both Jane and Sarah can access Mary's database and be recognized as its owner.

Dropping Aliases: *sp_dropalias*

The system procedure **sp_dropalias** drops the mapping of an alternate *suid* to a user ID, deleting the relevant row from *sysalternates*. Its syntax is:

```
sp_dropalias loginame
```

where *loginame* is the name of the user who was mapped to another user. Once a user's alias is dropped, the user no longer has access to the database, unless his or her name is then added to *sysusers* (with **sp_adduser**) or the database has a "guest" user in *sysusers*.

Getting Information About Aliases

To display information about aliases, use the **sp_helpuser** procedure. For example, to find the aliases for "dbo", execute:

```
sp_helpuser dbo
```

```
Users_name      ID_in_db    Group_name   Login_name   Default_db
----------      --------    ----------   ----------   ---------
dbo             1           public       sa           master

(1 row affected)
```

```
Users aliased to user.
Login_name
----------------------
andy
christa
howard
linda

(4 rows affected)
```

Getting Information About Users

The following procedures allow users to obtain information about users, groups and current SQL Server usage.

Table 4-6: System procedures that report on SQL Server users and groups

Procedure	Function
sp_who	Reports on current SQL Server users and processes
sp_displaylogin	Displays information about login accounts
sp_helpuser	Reports on users and aliases in a database
sp_helpgroup	Reports on groups within a database

Getting Reports on SQL Server Users and Processes: *sp_who*

The system procedure **sp_who** reports information about current users and processes on SQL Server. Its syntax is:

sp_who [*loginame* | "*spid*"]

- *loginame* is the user's SQL Server login name. If you give a login name, **sp_who** reports information about processes being run by the specified user.
- *spid* is the number of a specific process. Enclose it in quotes because a character type argument is expected.

For each process being run, **sp_who** reports the server process ID, its status, the login name of the process user, the name of the host computer, the server process ID of a process that's blocking this one (if any), the name of the database, and the command being run.

If you do not give a login name or *spid*, **sp_who** reports on processes being run by all users.

Here is an example of the results of executing **sp_who** without a parameter:

```
spid   status   loginame hostname blk dbname cmd
------ -------- -------- -------- --- ------ ----------------
     1 running  sa       sunbird  0   pubs2  SELECT
     2 sleeping NULL              0   master NETWORK HANDLER
     3 sleeping NULL              0   master MIRROR HANDLER
     4 sleeping NULL              0   master AUDIT PROCESS
     5 sleeping NULL              0   master CHECKPOINT SLEEP

(5 rows affected, return status = 0)
```

For all system processes, **sp_who** reports NULL for the *loginame.*

Getting Information About Login Accounts: *sp_displaylogin*

sp_displaylogin displays information about a specified login account. The syntax is:

sp_displaylogin [*loginame*]

where *loginame* is the user login account about which you want information. Only a System Administrator or System Security Officer can use **sp_displaylogin** with the *loginame* parameter to get information about other logins. If you wish to display information about your own login, you do not need to specify *loginame.*

sp_displaylogin displays the following information about any login account:

Server user ID
Login name
Full name
Any roles granted
Whether the account is locked
Date that the password was last changed

sp_displaylogin displays all roles that have been granted to you, so that even if you have made a role inactive with the **set** command, it is displayed.

Getting Information About Database Users: *sp_helpuser*

The system procedure **sp_helpuser** reports information about authorized users of the current database. Its syntax is:

sp_helpuser [*name_in_db*]

The *name_in_db* parameter, the user's name in the current database, is optional. If you give a user's name, **sp_helpuser** reports information about that user. If you don't give a name, it reports information about all users.

The procedure reports the user's name in the database, the user ID, the group name, the user's login name, and the user's default database. The following example shows the results of executing **sp_helpuser** without a parameter in the database *pubs2:*

```
sp_helpuser

Users_name  ID_in_db  Group_name Login_name  Default_db
----------  --------  ---------- ----------  ----------
dbo         1         public     sa          master
marcy       4         public     marcy       pubs2
sandy       3         public     sandy       pubs2
judy        5         public     judy        pubs2
linda       6         public     linda       master
anne        2         public     anne        pubs2
jim         7         senioreng  jim         master

(7 rows affected)
```

Finding User Names and IDs

To find a user's server user ID or login name, use the system functions **suser_id** and **suser_name.**

Table 4-7: System functions suser_id and suser_name

Function	Argument	Result
suser_id	([“server_user_name”])	Server user ID
suser_name	([server_user_ID])	Server user name (login name)

The arguments for these system functions are optional. If you don't give one, SQL Server displays information about the current user. For example, here we find the server user ID for the user “sandy”.

```
select suser_id("sandy")

------
     3
```

Here a System Administrator whose login name is “mary” issues the commands without arguments:

```
select suser_name(), suser_id()
------------------------------ ------
mary                                4
```

To find a user's ID number or name inside a database, use the system functions **user_id** and **user_name**.

Table 4-8: System functions user_id and user_name

Function	Argument	Result
user_id	(["db_user_name"])	User ID
user_name	([db_user_ID])	User name

The arguments for these system functions are optional. If you don't give one, SQL Server displays information about the current user. For example:

```
select user_name(10)
select user_name( )
select user_id("joe")
```

Getting Information About Usage: Chargeback Accounting

When a user logs into SQL Server, the server begins accumulating CPU and I/O usage for that user. Using this information, SQL Server can report total usage for an individual or for all users. Information for each user is kept in the *syslogins* system table in the *master* database.

A System Administrator can configure the frequency with which usage statistics are updated by setting the configuration parameters **cpu accounting flush interval** and **i/o accounting flush interval**. When the user logs out of a SQL Server session or accumulates more CPU or I/O usage than the configured **cpu accounting flush interval** or **i/o accounting flush interval** value, the new totals are written to *master..syslogins*.

System Procedures for Reporting Current Usage Statistics

The System Administrator can use either of the system procedures **sp_reportstats** or **sp_clearstats** to get or clear current total usage data for individuals or for all users on a SQL Server. To get or clear one user's totals, use the SQL Server login as a parameter; to get or clear the total for all users, use **sp_reportstats** or **sp_clearstats** with no parameters.

Using *sp_reportstats*

sp_reportstats displays a report of current accounting totals for SQL Server users. It reports total CPU and total I/O, as well as the percentage of those resources used. It does not record statistics for processes with an *suid* of 1 (the "sa" login), checkpoint, network, and mirror handlers.

Using *sp_clearstats*

SQL Server continues to accumulate CPU and I/O statistics until you clear the totals from *syslogins* by running **sp_clearstats**. **sp_clearstats** initiates a new accounting interval for SQL Server users. It executes **sp_reportstats** to print out statistics for the previous period and clears *syslogins* of the accumulated CPU and I/O statistics.

Choose the length of your accounting interval by deciding how you wish to use the statistics at your site. To do monthly cross-department charging for percentage of SQL Server CPU and I/O usage, for example, the System Administrator would run **sp_clearstats** once a month.

For detailed information about these stored procedures, see "sp_reportstats" and "sp_clearstats" in the *SQL Server Reference Manual*.

Configuration Parameters for Chargeback Accounting

Two configuration parameters allow a System Administrator to decide how often accounting statistics are added to *syslogins*:

- **cpu accounting flush interval** specifies how many machine clock ticks to accumulate before adding to *syslogins*. To find out how many microseconds a tick is on your system, run this query while logged into SQL Server:

  ```
  select @@timeticks
  ```

- **i/o accounting flush interval** specifies how many read or write I/Os to accumulate before flushing to *syslogins*. I/O and CPU statistics for an individual user are written out whenever the user exits a SQL Server session or accumulates more CPU or I/O usage than these values.

The minimum value allowed for either parameter is 1; the maximum value allowed is 2,147,483,647. The default value for **cpu accounting**

flush interval is 200, and the default value for **i/o accounting flush interval** is 1000.

To set these values, use the **sp_configure** command:

```
sp_configure "cpu accounting flush interval", 600
```

or:

```
sp_configure "i/o accounting flush interval", 2000
```

5 Roles in SQL Server

Introduction

SQL Server users can be granted special operational and administrative **roles**. Roles provide individual accountability for users performing system administration and security-related tasks. Roles are granted to individual server login accounts, and actions performed by these users can be audited and attributed to them.

These roles are:

- System Administrator
- System Security Officer
- Operator

In addition, there are two kinds of object owners who have special status because of the objects they own. These ownership types are:

- Database owner
- Database object owner

All of these are discussed in the following sections.

System and Security Administration Roles

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

These roles are granted to individual users with the **sp_role** system procedure. See “Granting and Revoking Roles” on page 5-6 for more information.

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

The System Administrator

A **System Administrator** performs administrative tasks 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. It is important,

however, that the System Administrator's functions be centralized or very well coordinated.

System Administrator Tasks

System Administrator tasks include:

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

System Administrator Permissions

A System Administrator operates outside the discretionary access control (DAC) protection system—SQL Server does no DAC permission checking when a System Administrator accesses objects. Also, a System Administrator takes on the identity of Database Owner in any database he or she enters, including *master*, by assuming the user ID 1.

There are several commands and system procedures that only a System Administrator can issue and on which permissions cannot be transferred to other users. For detailed information on System Administrator permissions, see Chapter 6, "Managing User Permissions".

System Security Officer

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

- Creating server login accounts

- Granting and revoking the System Security Officer and Operator roles
- Changing the password of any account
- Setting the password expiration interval
- Managing the audit system

The System Security Officer can **access** any database but, in general, has no special permissions on database objects. An exception is the *sybsecurity* database where only a System Security Officer can access the *sysaudits* table. There are also several system procedures that only a System Security Officer can execute and on which permissions cannot be transferred to other users.

The 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

SQL Server recognizes two types of owners:

- Database owners
- Database object owners

Database Owner

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

A Database Owner logs into SQL Server using his or her assigned login name and password. In other databases, that owner is known by his or her regular user name; in his or her own database SQL Server recognizes the user as "**dbo**." When SQL Server is installed, the "sa" login is the Database Owner of the *master* database.

Database Owner Tasks

The owner of a database may:

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

Adding users to databases is discussed in Chapter 4, "Managing SQL Server Logins and Database Users." Granting permissions to users is discussed in Chapter 6, "Managing User Permissions."

Database Owner Permissions

The Database Owner does not automatically receive permissions on objects owned by other users. However, a Database Owner can impersonate other users in the database at any time with the **setuser** command, temporarily assuming their permissions. Using a combination of the **setuser** and **grant** commands, the Database Owner can acquire permissions on any object in the database.

➤ ***Note***

Because database owners are so powerful, plan carefully who should own the databases in your server and consider appropriate auditing of their database activity.

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 the particular type of object. There are no special login names or passwords for database object owners.

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 it. System Administrators also have all permissions on the object. The owner of an object must explicitly grant permissions to other users before they can access it. 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.

➤ ***Note***

When a database object is owned by someone other than the Database Owner, the user (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.

Managing Roles in SQL Server

When it is installed, SQL Server includes an account called "sa" that is automatically granted the System Administrator, System Security Officer, and Operator roles during installation. The "sa" user is the Database Owner of the system databases. There are two options for managing the System Administrator role:

- You can use the "sa" account to perform all system administration and security-related functions. Any user who knows the "sa" password can use the account, and any actions performed can only be traced to "sa"; it is not possible to determine the individual user who was logged in as "sa".
- To increase accountability:
  - After installing SQL Server, use the "sa" account to create new server logins for users who are to be granted the System Administrator and System Security Officer roles.
  - Still logged in as "sa", grant the correct roles to these users using **sp_role**.
  - A user who has been granted the System Security Officer role can then log into his or her own account, lock the "sa" account

(using **sp_locklogin**), and create logins for other server users (using **sp_addlogin**). For a discussion about how to do this, together with an example, see Chapter 3, "Getting Started After Installation."

➤ ***Note***

If you decide to lock the "sa" account, be sure to check all scripts that may contain the "sa" login name and password. These can include scripts that perform backups, run **bcp**, or perform **dbcc** checking. The scripts cannot run if they are meant to run as "sa" and the "sa" account is locked. Change the logins in those scripts to the name of a user with the correct role.

Granting and Revoking Roles

The System Security Officer can grant the System Security Officer and Operator roles to other users, and the System Administrator can grant the System Administrator role to other users. This is done with the **sp_role** system procedure. The syntax is:

```
sp_role {"grant" | "revoke"},
   {sa_role | sso_role | oper_role}, loginame
```

grant and **revoke** specify whether you are granting or revoking the role.

The roles are **sa_role**, **sso_role**, or **oper_role**. Only one role can be specified in each execution of **sp_role**.

loginame is the name of the user to whom the role is being granted or from whom it is being revoked.

For example, this command:

```
sp_role "grant", sa_role, arthur
```

grants the role of System Administrator to "arthur".

When you grant a role to a user, it takes effect automatically the next time the user logs into SQL Server. If the user is already logged in, the user can enable the role by using the **set role** command. For example, this command:

```
set role sa_role on
```

immediately enables the System Administrator role for the user.

You cannot revoke a role from a user while the user is logged in.

You cannot lock or drop the last remaining System Security Officer or System Administrator account. The system procedures **sp_droplogin**,

sp_locklogin, and **sp_role** ensure that there is always at least one unlocked login that possesses the System Security Officer role, and one that possesses the System Administrator role.

Turning Roles On and Off

When you log into SQL Server, all roles granted to you are automatically enabled. Use the **role** option of the **set** command to turn any of your roles off or back on again for your current session. The syntax is:

```
set role {sa_role | sso_role | oper_role} {on | off}
```

This can be useful if, for example, you have been granted the System Administrator role, which means that you assume the identity of Database Owner within any database that you use. If, however, you want to assume your "real" user identity, execute this command:

```
set role sa_role off
```

If you are granted a role during a session and wish to activate it immediately, use **set role** to turn it on.

Checking for Roles

The following sections describe commands and procedures that allow you to check roles granted to yourself or others.

Displaying Login Account Information

You can use the **sp_displaylogin** system procedure to display information about a login 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.

The *show_role* Function

The **show_role** function displays any roles that are currently enabled for your login session. The syntax is:

```
select show_role()
```

It returns NULL if you have no roles enabled.

Checking for Roles in Stored Procedures: the *proc_role* Function

Use **proc_role** within a stored procedure to guarantee that only users with a specific role can execute the procedure. While **grant execute** can also restrict execute permission on a stored procedure, users without the required role might inadvertently be granted permission to execute it. Only **proc_role** provides a fail-safe way to prevent inappropriate access to a particular stored procedure.

proc_role takes a string for the required role (**sso_role, sa_role,** or **oper_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
```

If role auditing is enabled, the **proc_role** function also performs auditing for the actions taken by the possessor of the role.

6 Managing User Permissions

Introduction

The SQL commands **grant** and **revoke** control SQL Server's discretionary access control system. **Discretionary access controls** (DAC) allow you to restrict access to objects and commands based on a user's identity or group membership. The controls are "discretionary" in the sense that a user with a certain access permission, such as an object owner, can choose to pass that access permission on to other users.

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 databases, to create objects within a database, 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 (as System Administrator, Database Owner, or database object owner), and by whether or not a particular user has been granted a permission with the option to grant that permission to other users.

To supplement and complement the **grant** and **revoke** commands, you can use views and stored procedures as a security mechanism. This is discussed in "Permissions on Views and Stored Procedures" on page 6-22.

Permission Summary

System Administrators operate outside of the discretionary access control system, and have access permissions on all objects at all times.

Database Owners do not automatically receive permissions on objects owned by other users, but they can impersonate other users in their database at any time with the **setuser** command, temporarily assuming their permissions. Or, a Database Owner can permanently acquire permission on a given object by assuming the identity of the

object owner with the **setuser** command and then issuing the appropriate **grant** or **revoke** statements. The **setuser** command is discussed in more detail later in this chapter.

For permissions that default to "public", no permission is required, that is, no **grant** or **revoke** statements need ever be written.

Table 6-1 summarizes the discretionary access protection system of SQL Server. The type of user to whom the command defaults is the "lowest" level of user to which the permission is automatically granted. This user can **grant** the permission to other users or **revoke** it from other users, if it is transferable.

This table does not include System Security Officers. The System Security Officer does not have any special permissions on commands and objects, only on certain system procedures.

Table 6-1: Summary of permissions on SQL commands

Statement	Defaults to					Can be granted/revoked		
	System Admin.	Operator	Database Owner	Object Owner	Public	Yes	No	n/a
alter database			•			(1)		
alter table				•			•	
begin transaction					•			•
checkpoint			•				•	
commit transaction					•			•
create database	•					•		
create default			•			•		
create index				•			•	
create procedure			•			•		
create rule			•			•		
create table			•		(2)	• (2)		
create trigger				•			•	
create view			•			•		

(1) Transferred with database ownership
(2) Public can create temporary tables, no permission required
(3) If a view, permission defaults to view owner
(4) Defaults to stored procedure owner
(5) Transferred with **select** permission
(6) Transferred with **update** permission
No means use of the command is always restricted
n/a means use of the command is never restricted

Table 6-1: Summary of permissions on SQL commands (continued)

Statement	Defaults to					Can be granted/revoked		
	System Admin.	Operator	Database Owner	Object Owner	Public	Yes	No	n/a
dbcc	Varies depending upon options. See **dbcc** in the *SQL Server Reference Manual*						•	
delete				• (3)		•		
disk init	•						•	
disk mirror	•						•	
disk refit	•						•	
disk reinit	•						•	
disk remirror	•						•	
disk unmirror	•						•	
drop (any object)				•			•	
dump database		•	•				•	
dump transaction		•	•				•	
execute				•(4)		•		
grant on object				•		•		
grant command			•			•		
insert				• (3)		•		
kill	•						•	
load database		•	•				•	
load transaction		•	•				•	
print					•			•
raiserror					•			•
readtext				•		(5)		
revoke on object				•			•	
revoke command			•				•	
rollback transaction					•			•
save transaction					•			•
select				• (3)		•		

(1) Transferred with database ownership
(2) Public can create temporary tables, no permission required
(3) If a view, permission defaults to view owner
(4) Defaults to stored procedure owner
(5) Transferred with **select** permission
(6) Transferred with **update** permission
No means use of the command is always restricted
n/a means use of the command is never restricted

Table 6-1: Summary of permissions on SQL commands (continued)

Statement	Defaults to					Can be granted/revoked		
	System Admin.	Operator	Database Owner	Object Owner	Public	Yes	No	n/a
set					•			•
setuser			•				•	
shutdown	•						•	
truncate table				•			•	
update				• (3)		•		
update statistics				•			•	
writetext				•		(6)		

(1) Transferred with database ownership
(2) Public can create temporary tables, no permission required
(3) If a view, permission defaults to view owner
(4) Defaults to stored procedure owner
(5) Transferred with **select** permission
(6) Transferred with **update** permission
No means use of the command is always restricted
n/a means use of the command is never restricted

Types of SQL Server Users and Their Privileges

SQL Server's discretionary access control system recognizes these types of users:

- System Administrators
- System Security Officers
- Operators
- Database owners
- Database object owners
- Other users (also known as "public")

System Administrator Privileges

System Administrators are special users who handle tasks that are not specific to applications and who work outside SQL Server's discretionary access control system.

The role of System Administrator is usually granted to individual SQL Server logins. This provides a high degree of individual

accountability because all actions taken by that user can be traced to his or her individual server user ID. If the server administration tasks at your site are few enough that they are performed by a single individual, you may instead choose to use the "sa" account that is installed with SQL Server. At installation, "sa" has permission to assume the System Administrator, System Security Officer, and Operator roles. Any user who knows the "sa" password can log into that account and assume the System Administrator role.

The fact that a System Administrator operates outside the protection system serves as a safety precaution. For example, if the Database Owner accidentally deletes all the entries in the *sysusers* table, the System Administrator can restore the table (provided, of course, the Database Owner is making regular backups). There are several commands that can be issued only by a System Administrator—they cannot be granted to any other user. They include **disk init**, **disk refit**, **disk reinit**, **shutdown**, **kill**, and the disk mirroring commands.

In addition, System Administrators participate in login management in that they are responsible for dropping logins and can lock and unlock logins. System Security Officers share login management responsibilities with System Administrators. System Security Officers are responsible for adding logins and can also lock and unlock logins.

Permissions for Creating Databases

Only a System Administrator can grant permission to use the **create database** command. The **grant** command for permission on **create database** must be issued from *master*. In many installations, the System Administrator maintains a monopoly on **create database** permission in order to centralize control of database placement and database device space allocation.

In such situations, a System Administrator creates new databases on behalf of other users, and then transfers ownership to the appropriate user. To create a database that is to be owned by another user, the System Administrator issues the **create database** command, switches to the new database with the **use** command, and then executes the system procedure **sp_changedbowner**. Alternatively, System Administrators can grant **create database** permission to other users.

System Security Officer Privileges

System Security Officers perform security-sensitive tasks in SQL Server. These tasks include granting the System Security Officer and Operator roles, administering the audit system, changing passwords, adding new logins, and locking and unlocking login accounts. System Administrators share login management responsibilities with System Security Officers. System Administrators are responsible for dropping logins, and can also lock and unlock logins.

The System Security Officer can **access** any database but, in general, has no special permissions on database objects. An exception is the *sybsecurity* database, where only a System Security Officer can access the *sysaudits* table. The reason that System Security Officers can access any database is to allow them to enable auditing in any database. There are also several system procedures that can be executed only by a System Security Officer—permission to execute the procedures cannot be transferred to other users.

System Security Officers can repair any damage inadvertently done to the protection system by a user. For example, if the Database Owner forgets his or her password, a System Security Officer can change the password to allow the Database Owner to log in.

Operator Privileges

Users who have been granted the Operator role can back up and restore databases on a server-wide basis without having to be the owner of each database. The Operator role allows a user to use these commands on any database:

dump database
dump transaction
load database
load transaction

Database Owners also have dump and load permissions, but only on the databases they own. System Administrators can dump and load all databases.

Database Owner Privileges

Database Owners and System Administrators are the only users who can grant object creation permissions to other users.

The owner of a database has full privileges to do anything inside that database, and must explicitly grant permissions to other users with the **grant** command.

Following is a list of permissions that are automatically granted to the owner of a database and cannot be transferred to other users:

checkpoint
dbcc
setuser
dump database
dump transaction
load database
load transaction
drop database
grant and **revoke** object creation permissions

Note that users with the Operator role can use **dump database**, **dump transaction**, **load database**, and **load transaction** on any database.

Database Owners can grant these permissions to other users:

create table
create default
create rule
create procedure
create view
grant and **revoke** permissions on system tables
grant and **revoke select, insert, delete, update, references,** and **execute** permissions on database objects

Permissions on System Tables

The installation script supplied by SQL Server sets permissions on the system tables in a user database so that all database users have **select** permission on them.

However, the default situation is that no users—including Database Owners—can modify the system tables directly. Instead, the system stored procedures supplied with SQL Server modify the system tables. This helps guarantee integrity.

SQL Server does provide a mechanism to permit ad hoc changes to system tables, however. A System Security Officer can use **sp_configure** to change the **allow updates to system tables** configuration parameter.

◆ ***WARNING!***

Updating certain fields in the system tables prevents SQL Server from running. Therefore, allowing direct updates to the system tables is not recommended. For information and strict guidelines for allowing system table updates, see the *System Administration Guide*.

Permissions on System Procedures

Since system procedures are stored in the *sybsystemprocs* database, their permissions are also set there.

Security-related system procedures can only be run by System Security Officers. Certain other system procedures can only be run by System Administrators.

Some of the system procedures can be run only by Database Owners. These procedures make sure that the user executing the procedure is the owner of the database from which they are being executed.

Other system procedures can be executed by any user who has been granted permission—but this permission must be granted in *sybsystemprocs*. In other words, a user must have permission to execute a system procedure in all databases, or in none of them.

Users not listed in *sybsystemprocs..sysusers* are treated as "guest" in *sybsystemprocs*, and are automatically granted permission on many of the system procedures. In order to deny a user permission on a system procedure, the System Administrator must add him or her to *sybsystemprocs..sysusers* and issue a **revoke** statement that applies to that procedure. The owner of a user database cannot directly control permissions on the system procedures from within his or her own database.

The *setuser* Command

A Database Owner can use the **setuser** command to "impersonate" another user's identity and permissions status in the current database. A Database Owner may find it convenient to use **setuser** in order to access an object owned by another user, to grant permissions on an object owned by another user, to create an object that will be owned by another user, or to temporarily take on the discretionary access control (DAC) permissions of another user for some other reason.

While the **setuser** command enables the Database Owner to automatically acquire another user's DAC permissions, the command does not affect the roles that have been granted.

setuser permission defaults to the Database Owner and cannot be transferred. The user being impersonated must be an authorized user of the database.

System Administrators can use **setuser** in order to create objects that will be owned by another user. However, since System Administrators operate outside the discretionary access control permissions system, they need not use **setuser** to acquire another user's permissions.

The syntax of the **setuser** command is:

```
setuser ["user_name"]
```

where *user_name* is the name of the user to be impersonated. The user must be a valid user in the database. To reestablish your original identity, give the **setuser** command with no name after it.

Here's how the Database Owner would grant Joe permission to read the *authors* table, which is owned by Mary:

```
setuser "mary"

grant select on authors to joe

setuser     /*re-establishes original identity*/
```

When the Database Owner uses the **setuser** command, SQL Server checks the permissions of the user being impersonated.

The **setuser** command remains in effect until another **setuser** command is given, or until the current database is changed with the **use** command.

Changing Database Ownership

Use the system procedure **sp_changedbowner** to change the ownership of a database. Often, System Administrators create the user databases, then give ownership to another user after some of the initial work is complete. Only the System Administrator can execute **sp_changedbowner**.

It is a good idea to transfer ownership before the user has been added to the database, and before the user has begun creating objects in the database. The new owner must already have a login name on SQL Server, but cannot be a user of the database, or have an alias in the

database. You may have to use **sp_dropuser** or **sp_dropalias** before you can change a database's ownership, and you may have to drop objects before you can drop the user.

Issue **sp_changedbowner** in the database whose ownership will be changed. The syntax is:

```
sp_changedbowner loginame [, true ]
```

The following example makes the user "albert" the owner of the current database and drops aliases of users who could act as the old "dbo":

```
sp_changedbowner albert
```

To transfer aliases and their permissions to the new "dbo," add the second parameter with the value "true" or "TRUE".

➤ ***Note***

You cannot change the ownership of the *master* database and should not change the ownership of any other system databases.

Database Object Owner Privileges

A user who creates a database object (a table, view, or stored procedure) owns the object and is automatically granted all object access permissions on it. Users other than the object owner, including the owner of the database, are automatically denied all permissions on that object, unless they are explicitly granted by either the owner or a user who has **grant** permission on that object.

As an example, suppose that Mary is the owner of the *pubs2* database, and has granted Joe permission to create tables in it. Now Joe creates the table *new_authors*; he is the owner of this database object.

Initially, object access permissions on *new_authors* belong to Joe and Joe alone. Joe can grant or revoke object access permissions for this table to other users.

These are the object creation permissions that default to the owner of a table, and that cannot be transferred to other users:

alter table
drop table
create index
create trigger

truncate table
update statistics

Permission to use the **grant** and **revoke** commands to grant specific users **select, insert, update, delete, references,** and **execute** permissions on specific database objects can be transferred, using the **grant with grant option** command.

Permission to **drop** an object—a table, view, index, stored procedure, rule, or default—defaults to the object owner and cannot be transferred.

Privileges of Other Database Users

At the bottom of the hierarchy are other database users. Permissions are granted to or revoked from them by object owners, Database Owners, users who were granted permissions with the grant option, or a System Administrator. These users are specified by user name, by group name, or by the keyword **public**.

Granting and Revoking Permissions

There are two categories of permissions on objects:

- Permissions for **select**, **update**, **insert**, **delete**, **references**, and **execute** are called **object access permissions** because they access database objects. The **references** permission refers to referential integrity constraints that you can specify in an **alter table** or **create table** command. The other permissions refer to SQL commands. Object access permissions are discussed in the next section, "Object Access Permissions" on page 6-12.
- Permissions to create objects are called **object creation permissions**. They can be granted only by a System Administrator or a Database Owner. These permissions are discussed in "Object Creation Permissions" on page 6-16.

Both types of permissions are controlled with the **grant** and **revoke** commands.

Each database has its own independent protection system: having permission to use a certain command in one database does not give you permission to use that command in other databases.

If you try to use a command or database object for which you have not been assigned permission, SQL Server displays an error message.

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 access permissions are granted and revoked by the object owner, who can grant them to other users.

Table 6-2 lists the types of object access permissions and the objects to which they apply:

Table 6-2: 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

Object access permissions default to System Administrators and the object's owner, and can be granted to other users.

grant and *revoke* Syntax: Object Access Permissions

Here are the syntax statements for granting and revoking object access permissions (permission to use tables, views, columns, and stored procedures):

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

Notes on the keywords and parameters are as follows:

- **all** or **all privileges** specifies that all permissions applicable to the specified object are granted or revoked. All object owners can use

all with an object name to grant or revoke permissions on their own objects. If you are granting or revoking permissions on a stored procedure, **all** is the same as **execute**.

➤ ***Note***

insert and **delete** permissions do not apply to columns, so you cannot include them in a permission list (or use the keyword **all**) if you specify a column list.

- *permission_list* is the list of permissions that you are granting. If you name more than one permission, separate them with commas. The following table illustrates the access permissions that can be granted on each type of object:

Table 6-3: Object access permissions

Object	*permission_list* Can Include:
Table or view	**select**, **insert**, **delete**, **update**, **references.** **references** applies to tables but not views; the other permissions apply to both tables and views.
Column	**select**, **update**, **references**
Stored procedure	**execute**

You can specify columns either in the *permission_list* or in the *column_list*, but not both.

- **on** specifies the object for which the permission is being granted or revoked. You can grant or revoke permissions for only one table, view, or stored procedure object at a time. You can grant or revoke permissions for more than one column at a time, but all the columns must be in the same table or view. You can only grant or revoke permissions on objects in your current database.
- **public** refers to the group "public", which includes all the users of SQL 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. For **revoke** on object creation permissions, **public** excludes the Database Owner (who "owns" object creation permissions).

- *name_list* is a list of the names of:
  - Groups
  - Users
  - A combination of users and groups, each name separated from the next by a comma
- *role_name* is the name of a SQL 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).
- **with grant option** in a **grant** statement allows the user(s) specified in *name_list* to 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 a group of which the user is a member.
- The **grant option for** revokes **with grant option** permissions, so that the user(s) specified in *name_list* can no longer grant the specified permissions to other users. If those other users have granted permissions to other users, you must use the **cascade** option to revoke permissions from them as well. The user specified in *name_list* retains permission to access the object, but can no longer grant access to other users. **grant option for** applies only to object access permissions, not to object creation permissions.
- 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.

You may only grant and revoke permissions on objects in the current database.

Permissions granted to roles override permissions granted to users or groups. For example, suppose that 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 can still access *sales* because his role permissions override his individual permissions.

Special Requirements for Compliance to 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 6-4: 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 column owner must grant you **select** permission on all relevant columns.

Examples: Granting and Revoking Object Access Permissions

This 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
```

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

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

This 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
```

This 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
```

Both of the following statements revoke permission for 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
```

This 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
```

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

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

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 object creation commands are:

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

grant and *revoke* Syntax: Object Creation Permissions

The syntax for object creation permissions differs slightly from the syntax for object access permissions. Here are the syntax statements for object creation permissions:

```
grant {all [privileges] | command_list}
   to {public | name_list | role_name}

revoke {all [privileges] | command_list}
   from {public | name_list | role_name}
```

Notes on the keywords and parameters are as follows:

- **all** or **all privileges** 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 the Database Owner uses **grant all**, SQL Server grants all **create** permissions except **create database**, and prints an informational message.
- 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 only be granted by a System Administrator, and only from within the *master* database.
- **public** is all users. For object creation permissions, **public** excludes the Database Owner (who "owns" object creation permissions within the database).
- *name_list* is a list of user or group names, separated by commas.
- *role_name* is the name of a SQL Server role. This allows you to grant specific 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). Roles are granted to users with **sp_role**.

Examples: Granting and Revoking Object Creation Permissions

Here are a few examples that show how to grant and revoke object creation permissions.

The first example grants Mary and John permission to use the **create database** and **create table** commands. Because **create database** permission is being granted, this command can only be executed by a System Administrator within the *master* database. Mary and John's **create table** permission applies only to the *master* database.

```
grant create table, create database
to mary, john
```

The following command grants permission to create tables and views in the current database to all users:

```
grant create table, create view
to public
```

The following example revokes permission to create tables and rules from "mary":

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

Combining *grant* and *revoke* Statements

There are two basic styles of setting up permissions in a database or on a database object. The most straightforward 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
```

Conflicting *grant* and *revoke* Statements

The **grant** and **revoke** statements are sensitive to the order in which they are issued. So, for example, if Joe's group has been granted **select** permission on the *titles* table and then Joe's permission to select the *advance* column has been revoked, Joe can select all the columns except *advance*, while the other users in his group can still select all the columns.

A **grant** or **revoke** command that applies to a group changes any conflicting permissions that have been assigned to any member of that group. For example, suppose that the owner of the *titles* table has granted different permissions to various members of the *sales* group, and then decides 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** changes, for all users, any 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 Joe without any **select** permission on *titles*:

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

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

Remember that 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 wish 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 always change your mind and reinstitute the permission with a **grant** statement.)

Reporting on Permissions

These system procedures provide information about object creation and object access permissions:

- **sp_helprotect** reports on permissions on database objects or users.
- **sp_column_privileges** reports on permissions on specific columns in a table.
- **sp_table_privileges** reports on permissions on a specific table.

sp_helprotect

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. Its syntax is:

```
sp_helprotect name [, username [, "grant"]]
```

The first parameter, *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 the second parameter, *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 so, the permissions for the user, group, or role are listed. If you specify the third parameter, the keyword **grant**, and *name* is not an object, **sp_helprotect** displays all permissions granted by **with grant option.**

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, type:

```
sp_helprotect titles, judy
```

The following display results:

```
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 row of the display shows that the Database Owner ("dbo") gave Judy permission to select all columns of the *titles* table. The rest of the lines indicate that she can update only the columns listed in the display. Judy's permissions are not grantable: she cannot give **select** or **update** permissions to any other user.

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

```
sp_helprotect publishers, judy
```

In the display below, the *grantable* column indicates 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
```

sp_column_privileges

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

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

Following is a description of the parameters:

- *table_name* is the name of the table. This is required.
- *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 want to skip.

For example, this statement:

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

returns information about the *pub_id* column of the *publishers* table. See the *SQL Server Reference Manual* for more specific information about the output of **sp_column_privileges**.

sp_table_privileges

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

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

Use **null** for parameters that you want to skip.

Following is a description of the parameters:

- *table_name* is the name of the table. It is required.
- *table_owner* can be used to specify the name of the table owner, if it is not "dbo" or the user executing **sp_table_privileges**.
- *table_qualifier* is the name of the current database.

For example, this statement:

```
sp_table_privileges titles
```

returns information about all permissions granted on the *titles* table. See the *SQL Server Reference Manual* for more specific information about the output of **sp_table_privileges**.

Permissions on Views and Stored Procedures

Views and stored procedures can serve as security mechanisms. You can give users controlled access to database objects via a view or stored procedure without granting them direct access to the data. For example, you might give a clerk **execute** permission on a procedure that updates cost information in a *projects* table without letting the user see confidential data in the table. To take advantage of this feature, you must own the procedure or view as well as its underlying objects. If you do not own the underlying objects, users must have permission to access the objects. For more information about when permissions are required, see "Ownership Chains" on page 6-26.

SQL Server makes permission checks, as required, when the view or procedure is used. When you create the view or procedure, SQL Server makes no permission checks on the underlying objects. One exception to this occurs when you try to create a procedure that accesses objects in another database. If you do not have the required permissions for the database objects in the other database, SQL Server gives you an error message, and the procedure creation fails.

Views as Security Mechanisms

Through a view, users can query and modify only the data they can see. The rest of the database is neither visible nor accessible.

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. If the view and underlying tables are owned by the same owner, no permissions need to be given to the underlying tables. 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.

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.

As an example, 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, and then give all users permission on the view but only the Sales Department permission on the table. Here's how:

```
grant all on bookview to public

grant all on titles to sales
```

An equivalent way of setting up these privilege conditions, without using a view, is this series of statements:

```
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 scheme 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
```

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

In order 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.

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. In order to do so, 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, like this:

```
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 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.

Stored Procedures as Security Mechanisms

If a stored procedure and all underlying objects are owned by the same user, that owner can grant users permission to use the procedure without granting permissions on the underlying objects. 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 other permissions on that table.

Roles and Stored Procedures

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. But **grant execute** permission does not prevent users who do **not** have the specified role from being granted execute permission on the stored procedure.

For further security, you can restrict the use of a stored procedure by using the **proc_role** system function within the procedure to guarantee that a procedure can be executed only by users who have a given

role. **proc_role** returns 1 if the user has a specific role (**sa_role**, **sso_role**, or **oper_role**) and returns 0 if the user does not have that role. For example, here is a procedure that uses **proc_role** to see if the user has the System Administrator 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
```

See "System Functions" in the *SQL Server Reference Manual* for more information about **proc_role**.

Ownership Chains

Views can depend on other views and / or tables. Procedures can depend on other procedures, views, and / or tables. These dependencies can be thought of as an **ownership chain**.

Typically, the owner of a view also owns its underlying objects (other views and tables), and the owner of a stored procedure owns all the procedures, tables, and views referenced by the procedure.

Also, a view and its underlying objects are usually all in the same database, as are a stored procedure and all the objects it references. However, it is not required that all of the underlying objects be in the same database. If these objects are in different databases, a user wishing to use the view or stored procedure must be a valid user or guest user in all of the databases containing the objects. (This prevents users from accessing a database unless the Database Owner has authorized it.)

When a user who has been granted **execute** permission on a procedure or view uses it, SQL Server does not check permissions on any of the underlying objects if:

- These objects and the view or procedure are owned by the same user, and
- The user accessing the view or procedure is a valid user or guest user in each of the databases containing the underlying objects.

However, if all objects are not owned by the same user, SQL Server checks object permissions when the ownership chain is broken. That is, if object A references object B, and B is not owned by the user who owns object A, SQL Server checks the permissions for object B. In this way, SQL Server allows the owner of the original data to retain control over who is authorized to access it.

Ordinarily, a user who creates a view need worry only about granting permissions on that view. For example, say Mary has created a view called *auview1* on the *authors* table, which she also owns. If Mary grants **select** permission to Sue on *auview1*, SQL Server will let Sue access it without checking permissions on *authors*.

However, a user who creates a view or stored procedure that depends on an object owned by another user must be aware that any permissions he or she grants depend on the permissions allowed by those other owners.

Example: Views and Ownership Chains

Say Joe creates a view called *auview2*, which depends on Mary's view *auview1*. Joe grants Sue **select** permission on *auview2*.

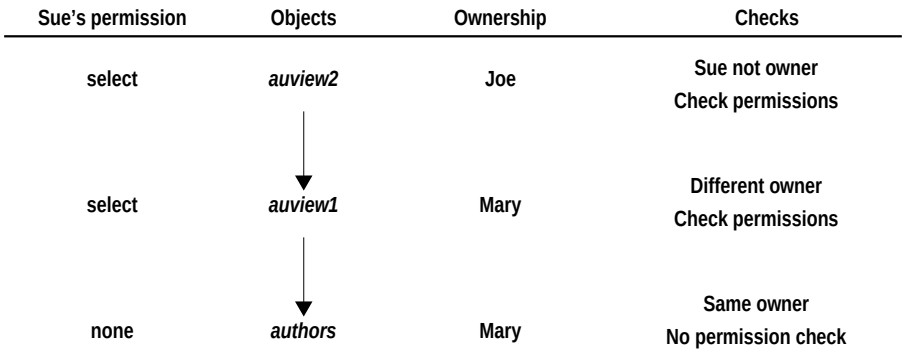

Figure 6-1: Ownership chains and permission checking for views, case 1

SQL Server checks the permissions on *auview2* and *auview1*, and finds that Sue can use them. SQL Server checks ownership on *auview1* and *authors* and finds that they have the same owner. Therefore, Sue can use *auview2*.

Taking this example a step further, suppose that Joe's view, *auview2*, depends on *auview1*, which depends on *authors*. Mary decides she

likes Joe's *auview2* and creates *auview3* on top of it. Both *auview1* and *authors* are owned by Mary.

The ownership chain looks like this:

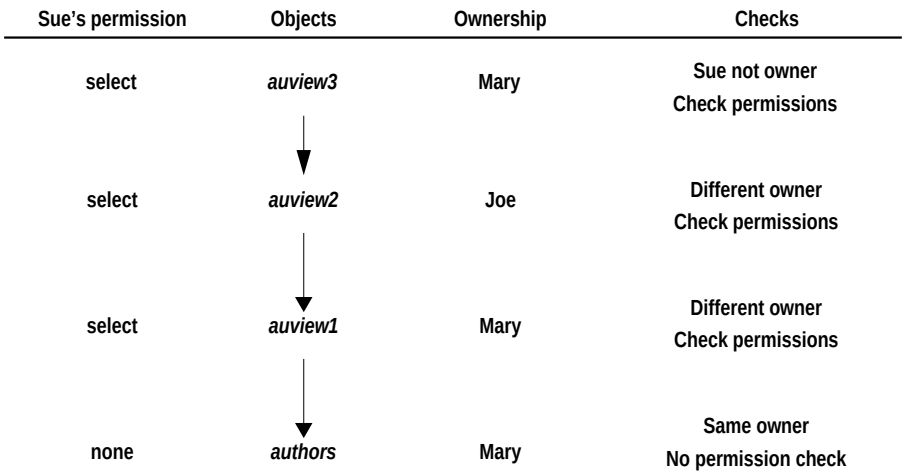

Figure 6-2: Ownership chains and permission checking for views, case 2

When Sue tries to access *auview3*, SQL Server checks permissions on *auview3*, *auview2*, and *auview1*. If Joe has granted permission to Sue on *auview2* and Mary has granted her permission on *auview3* and *auview1*, SQL Server allows the access. SQL Server checks permissions only if the object immediately before it in the chain has a different owner (or if it is the first object in the chain). For example, it checks *auview2* because the object before it—*auview3*—is owned by a different user. It does not check permission on *authors*, because the object which immediately depends on it, *auview1*, is owned by the same user.

Example: Procedures and Ownership Chains

Procedures follow the same rules as views. For example, suppose the ownership chain looks like this:

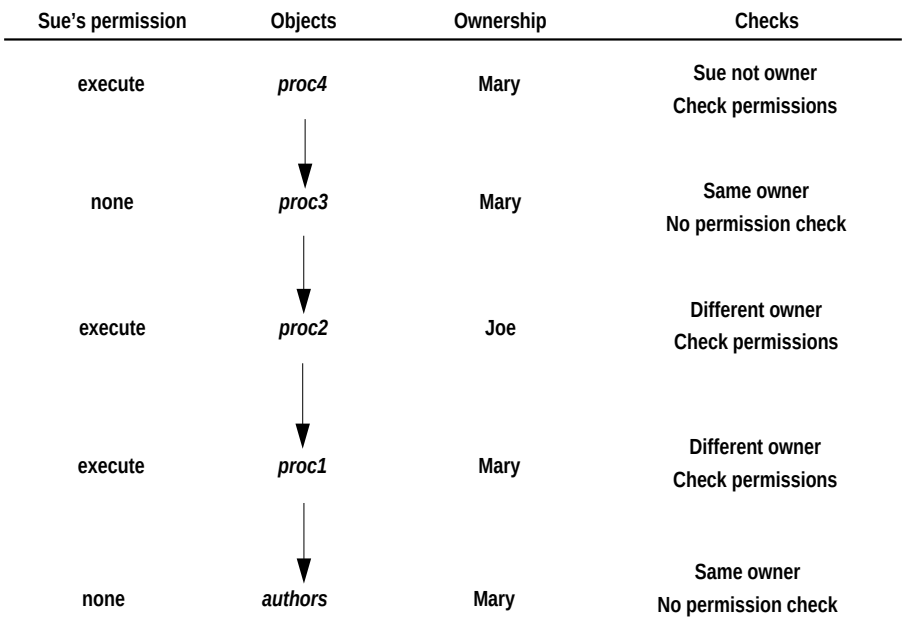

Figure 6-3: Ownership chains and permission checking for stored procedures

To execute *proc4*, Sue must have permission to execute *proc4*, *proc2*, and *proc1*. Permission to execute *proc3* is not necessary, since *proc3* and *proc4* have the same owner.

SQL Server checks Sue's permissions on *proc4* and all objects it references each time she executes *proc4*. SQL Server knows which referenced objects to check: it determined this the first time Sue executed *proc4*, and it saved the information with the procedure's execution plan. Unless one of the objects referenced by the procedure is dropped or otherwise redefined, SQL Server does not change its initial decision about which objects to check.

The purpose of this protection hierarchy is to allow every object's owner to fully control access to the object. Owners can control access to views and stored procedures, as well as to tables.

Triggers

A **trigger** is a special kind of stored procedure used to enforce integrity, especially referential integrity. (See the *Transact-SQL User's*

Guide or the *SQL Server Reference Manual* for details.) Triggers are never executed directly, but only as a side effect of modifying a table. There is no way to **grant** or **revoke** permissions for triggers.

Only the owner of an object can create a trigger on it. However, the ownership chain can be broken if a trigger on a table references objects owned by different users. The protection hierarchy rules that apply to procedures also apply to triggers.

While the objects that a trigger affects are usually owned by the user who owns the trigger, you can write a trigger that modifies an object owned by another user. If this is the case, any users modifying your object in a way that activates the trigger must have permission on the other object as well.

If SQL Server denies permission on a data modification command because a trigger affects an object for which the user does not have permission, the entire data modification transaction is rolled back.

7 Managing Remote Servers

Users on a local SQL Server can execute stored procedures on a remote SQL Server. Executing a remote procedure call sends the results of the remote process to the calling process—usually the user's screen. This chapter discusses the steps the System Administrator and System Security Officer of each SQL Server must execute to enable remote procedure calls. They are:

- On the local server:
  - Both the local server and remote server must be listed in the system table *master..sysservers* (with **sp_addserver**). This must be done by a System Security Officer.
  - The remote server must be listed in the interfaces file for the local server. (The interfaces file, set up when SQL Server is installed, lists the names and addresses of all the servers you can access.)
- On the remote server:
  - The server originating the remote procedure call must be listed in the *sysservers* table (with **sp_addserver**). This must be done by a System Security Officer.
  - The user originating the remote procedure must be allowed access to the server (with **sp_addlogin** and **sp_addremotelogin**). **sp_addlogin** is executed by a System Security Officer, **sp_addremotelogin** by a System Administrator.
  - The remote login name must be added as a user of the appropriate database and must have permission to execute the procedure. (If **execute** permission is granted to "public", the user does not need to be granted specific permission.)

Figure 7-1 provides an example of setting up servers for remote access.

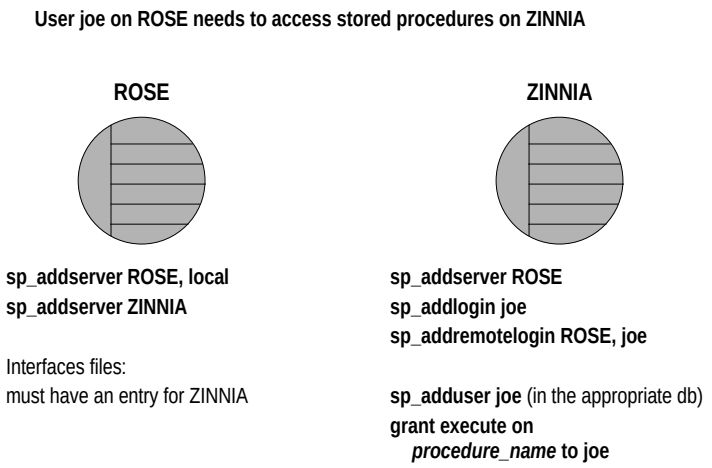

Figure 7-1: Setting up servers to allow remote procedure calls

For operating system-specific information about handling remote servers, see the SQL Server installation and configuration guide.

Managing Remote Servers

Four system procedures are used to manage remote servers:

- **sp_addserver** – adds a server name to *master..sysservers*
- **sp_dropserver** – drops a server name from *master..sysservers*
- **sp_helpserver** – displays information about servers
- **sp_serveroption** – displays or changes server options

Adding a Remote Server

The system procedure **sp_addserver** adds entries to the *sysservers* table. On the server originating the call, you must add an entry for the local server, and an entry for each remote server that your server will call. Only a System Security Officer can execute **sp_addserver**. When you create entries for a remote server, you can choose to:

- Always refer to them by the name listed in the interfaces file, or
- Provide a local name for the remote server. For example, if the name in the interfaces file is "MAIN_PRODUCTION", you may want to call it simply "main".

Here is the syntax:

```
sp_addserver lname [{, local | null}
   [, pname]]
```

Following is a description of each parameter you can specify for **sp_addserver**:

- The *lname* parameter provides the local "call name" for the remote server. Users will type this name when they issue remote procedure calls. If this name is **not** the same as the remote server's name in the interfaces file, you must provide that name as the third parameter, *pname*.

  The remote server must be listed in the interfaces file on the local machine. If it's not listed, copy the interfaces file entry from the remote server and append it to your existing interfaces file. Be sure to keep the same port numbers.
- The **local** option identifies the server being added as a local server. The **local** value is used only after start-up, or after a reboot, to identify the local server name so that it can appear in messages printed out by SQL Server. **null** specifies that this server is a remote server.
- For *pname*, give the name for the remote server that is listed in the interfaces file for the server named *lname*. This optional third argument permits you to establish local aliases for any other SQL Server, Open Server™, or Backup Server that you may need to communicate with. If you do not specify a *pname*, it defaults to *lname*.

The server originating a remote procedure call must have a local entry. You must restart SQL Server to set the value of the global variable *@@servername*.

Examples of Using *sp_addserver*

The following example creates an entry for the local server named DOCS:

```
sp_addserver DOCS, local
```

This example creates an entry for a remote server named GATEWAY:

```
sp_addserver GATEWAY
```

If you want to run a remote procedure such as **sp_who** on the GATEWAY server, execute:

```
GATEWAY.sybsytemprocs.dbo.sp_who
```

Or:

```
GATEWAY...sp_who
```

The following example gives a remote server called MAIN_PRODUCTION the local alias "main":

```
sp_addserver main, null, MAIN_PRODUCTION
```

Users can then type:

```
main...sp_who
```

Managing Remote Server Names

The *master.dbo.sysservers* table has two name columns:

- *srvname* is the name that users must supply when executing remote procedure calls. *srvname* must be unique on each server.
- *srvnetname* is the server's network name, which must match the name in the *interfaces* file. *srvnetname* does not have to be unique; you can have more than one local name for a remote server.

If you need to add or drop servers from your network, you can use **sp_addserver** to update the *srvnetname*. If you need to remove the server MAIN_PRODUCTION from the network, and move your remote applications to TEMP_PRODUCTION, here's how to change the network name, while keeping the local alias:

```
sp_addserver main, null, TEMP_PRODUCTION
```

The **sp_addserver** procedure prints an informational message telling you that it is changing the network name of an existing server entry.

Dropping Remote Servers: *sp_dropserver*

The **sp_dropserver** system procedure drops servers from *sysservers*. The syntax is:

```
sp_dropserver server [, droplogins]
```

where *server* is the name of the server you want to drop. Only System Security Officers can execute **sp_dropserver.** The **droplogins** option allows you to drop a remote server and all of that server's remote login information in one step. If you don't use the **droplogins** option,

you cannot drop a server that has remote logins associated with it. The following statement drops the GATEWAY server and all of the remote logins associated with it:

```
sp_dropserver GATEWAY, droplogins
```

The **droplogins** option isn't needed if you wish to drop the local server; that entry won't have remote login information associated with it.

Setting Server Options: *sp_serveroption*

The **sp_serveroption** system procedure sets the server options **timeouts** and **net password encryption**, which affect connections with remote servers. These options do not affect SQL Server to Backup Server communication.

The *timeouts* Option

timeouts disables and enables the normal timeout code used by the local server, so the site connection handler does not automatically drop the physical connection after a minute with no logical connection. This option can be set only by a System Administrator.

By default, **timeouts** is set to TRUE, and the site handler process that manages remote logins times out if there has been no remote user activity for one minute. By setting **timeouts** to FALSE on both of the servers involved in remote procedure calls, the automatic timeout is disabled. This example changes the **timeouts** option to FALSE:

```
sp_serveroption GATEWAY, "timeouts", false
```

Once you set **timeouts** to FALSE on both servers and a user executes a remote procedure call in either direction, the site handler on each machine runs until one of the servers is shut down. (When the server is brought up again, the option remains FALSE, and the site handler will be reestablished the next time a user executes a remote procedure call.) If users on the server execute remote procedure calls frequently, it is probably efficient in terms of system resources to set this option to FALSE, since there is some system overhead involved in setting up the physical connection.

The *net password encryption* Option

net password encryption specifies whether connections with a remote server are to be initiated with a client-side password encryption handshake or with the normal (unencrypted password) handshake

sequence. The default is FALSE. This option can be set only by a System Security Officer.

If **net password encryption** is set to TRUE, the following occurs:

1. The initial login packet is sent without passwords.
2. The client indicates to the remote server that encryption is desired.
3. The remote server sends back an encryption key, which the client uses to encrypt its plaintext passwords.
4. The client then encrypts its own passwords, and the remote server uses the key to authenticate them when they arrive.

This example sets this option to TRUE:

```
sp_serveroption GATEWAY, "net password encryption",
    true
```

Setting this option has no effect on SQL Server's interaction with Backup Server.

Getting Information About Servers: *sp_helpserver*

The system procedure **sp_helpserver** reports on servers. When it is used without an argument, it provides information on all the servers listed in *sysservers*. It can take a server name as an argument, and provides information on just that server.

```
sp_helpserver [server]
```

sp_helpserver checks for both *srvname* and *srvnetname* in the *master..sysremotelogins* table.

For operating system specific information about setting up remote servers, see the SQL Server installation and configuration guide, which contains detailed examples of setting up your environment for remote procedure calls.

Adding Remote Logins

The System Security Officer and System Administrator of any SQL Server share control over which remote users can access the server, and what identity the remote users assume. The System Administrator uses **sp_addremotelogin** to add remote logins and **sp_dropremotelogin** to drop remote logins. The System Security Officer

uses **sp_remoteoption** to control whether password checking will be required.

Mapping Users' Server IDs: *sp_addremotelogin*

Logins from a remote server can be mapped to a local server in three ways:

- A particular remote login can be mapped to a particular local login name. For example, the user "joe" on the remote server might be mapped to "joesmith", or user "judy" to "judyj".
- All logins from one remote server can be mapped to one local name. For example, all users sending remote procedure calls from the MAIN_PRODUCTION server might be mapped to "remusers".
- All logins from one remote server can use their remote names.

The first option can be combined with the other two options, and its particular mapping takes precedence over the other two more general mappings.

The second and third options are mutually exclusive; you may use either of them, but not both. To change from one of these options to the other, you must use **sp_dropremotelogin** to remove the old mapping. For information about **sp_dropremotelogin**, see the *SQL Server Reference Manual*.

The system procedure **sp_addremotelogin** adds remote logins. The syntax is:

```
sp_addremotelogin remoteserver [, loginame
   [, remotename]]
```

Only a System Administrator can execute **sp_addremotelogin**. If the local names are not listed in *master..syslogins*, you must first add them as SQL Server logins with **sp_addlogin**.

Mapping Remote Logins to Particular Local Names

The following example maps the login named "pogo" from a remote system to the local login name "bob". The user logs into the remote system as "pogo"; whenever that user executes remote procedure calls from GATEWAY, the local system maps the remote login name to "bob".

```
sp_addlogin bob
```

```
sp_addremotelogin GATEWAY, bob, pogo
```

Mapping All Remote Logins to One Local Name

The following example creates an entry that maps all remote login names to the local name "albert". All names are mapped to "albert", except those with specific mapping, as described above. For example, if you mapped "pogo" to "bob", and then the rest of the logins to "albert", "pogo" still maps to "bob".

```
sp_addlogin albert
sp_addremotelogin GATEWAY, albert
```

◆ ***WARNING!***

Mapping more than one remote login to a single local login is not recommended, as it reduces individual accountability on the server. Audited actions can be traced only to the local server login, not to the individual logins on the remote server.

Remote Logins Keeping Remote Names

If you want remote users to keep their remote login names while using a local server, first use **sp_addlogin** to create a login for each login from the remote server. Then, use **sp_addremotelogin** for the server as a whole:

```
sp_addremotelogin GATEWAY
```

This command creates an entry in *master..sysremotelogins* with a null value for the remote login name, and a value of -1 for the *suid*.

An Example of Remote User Mapping

Assume that you have the following servers listed in *master..sysservers* on the SALES server:

```
select srvid, srvname from sysservers
```

```
srvid  srvname
-----  ----------
    0  SALES
    1  CORPORATE
    2  MARKETING
    3  PUBLICATIONS
    4  ENGINEERING
```

```
select remoteserverid, remoteusername, suid
  from sysremotelogins
```

and the following entries in *master..sysremotelogins*:

```
remoteserverid   remoteusername   suid
--------------   --------------   ------
    1                joel             2
    1                freddy           2
    1                NULL             3
    3                NULL             4
    4                NULL             -1
```

and these entries in *master..syslogins*:

```
select suid, name from syslogins
```

```
suid    name
------  ------------
    1   sa
    2   vp
    3   peon
    4   writer
```

The effect is:

- The logins “joel” and “freddie” from the CORPORATE server are known as “vp”.
- All other logins from CORPORATE are mapped to the name “peon”.
- All logins from PUBLICATIONS are mapped to “writer”.
- All logins from ENGINEERING are looked up in *master..syslogins* by their remote user names.
- Users from MARKETING cannot run remote procedure calls on this server because there is no entry for MARKETING in *sysremotelogins*.

The remote user mapping procedures and the ability to set permissions for individual stored procedures give you control over which remote users can access local procedures. For example, you can allow the “vp” login from CORPORATE to execute certain local

procedures and all other logins from CORPORATE to execute the procedures for which "peon" has permission.

➤ ***Note***

In many cases, the passwords for users on the remote server must match passwords on the local server. See the next section for details.

Password Checking for Remote Users: *sp_remoteoption*

The system procedure **sp_remoteoption** determines whether passwords will be checked when remote users log into the local server. By default, passwords are verified ("untrusted" mode). In **trusted** mode, the local server accepts remote logins from other servers and front-end applications without user-access verification for the particular login.

◆ ***WARNING!***

Using the trusted mode of sp_remoteoption reduces the security of your server, as passwords from such "trusted" users are not verified.

When **sp_remoteoption** is used with arguments, it changes the mode for the named user. Here is the syntax:

```
sp_remoteoption [remoteserver, loginame, remotename,
   optname, {true | false}]
```

Only a System Security Officer can execute **sp_remoteoption.**

This example sets **trusted** mode for the user "bob":

```
sp_remoteoption GATEWAY, pogo, bob, trusted,
    true
```

The effects of the "untrusted" mode depend on the user's client program. **isql** and some user applications require that logins have the same password on the remote server and the local server. Open Client™ applications can be written to allow local logins to have different passwords on different servers.

To change your password in "untrusted" mode, you must first change it on all the remote systems you access before changing it on your local server. This is because of the password checking. If you change your password on the local server first, when you issue the

remote procedure call to execute **sp_password** on the remote server your passwords will no longer match.

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

```
remote_server...sp_password caller_passwd, new_passwd
```

And on the local server:

```
sp_password caller_passwd, new_passwd
```

See "Changing Passwords: sp_password" on page 4-14 for more information about changing your password.

Getting Information About Remote Logins

The stored procedure **sp_helpremotelogin** prints information about the remote logins on a server. The following example shows the remote login "pogo" mapped locally to login name "bob", with all other remote logins keeping their remote names.

```
sp_helpremotelogin
server      remote_user_name      local_user_name       options
-------     ----------------      -----------------     --------
GATEWAY     **mapped locally**    **use local name**    untrusted
GATEWAY     pogo                  bob                   untrusted
```

Configuration Parameters for Remote Logins

Each SQL Server that allows remote logins to execute procedure calls must have certain configuration parameters set. These values are changed with **sp_configure**.

The values that affect remote procedure calls are **allow remote access**, **number of remote logins**, **number of remote sites**, **number of remote connections**, and **remote server pre-read packets**. All these configuration parameters are static and do not take effect until SQL Server is rebooted.

By default, **allow remote access** is set to 1 to allow remote access. This parameter must be set to 1 to allow communication with Backup Server. The other values are set to reasonable defaults, as shown in Table 7-1.

Table 7-1: Configuration parameters that affect RPCs

Configuration Parameter	Default
allow remote access	1
number of remote logins	20

Table 7-1: Configuration parameters that affect RPCs

Configuration Parameter	Default
number of remote sites	10
number of remote connections	20
remote server pre-read packets	3

The **allow remote access** configuration parameter can be changed only by a System Security Officer. The other parameters can be changed only by a System Administrator. All the parameters are described in the following sections.

allow remote access

The **allow remote access** parameter must be set to 1 to allow remote access to or from a server, including access to Backup Server:

```
sp_configure "allow remote access", 1
```

To disallow remote access at any time, reset the **allow remote access** parameter to 0:

```
sp_configure "allow remote access", 0
```

Only a System Security Officer can set the **allow remote access** parameter.

◆ ***WARNING!***

You cannot perform database or transaction log dumps while the allow remote access configuration parameter is set to 0.

number of remote logins

The **number of remote logins** parameter controls the number of active user connections from this site to remote servers. This command sets **number of remote logins** to 50:

```
sp_configure "number of remote logins", 50
```

Only a System Administrator can set the **number of remote logins** parameter.

number of remote sites

The **number of remote sites** parameter controls the number of remote sites that can access this server simultaneously. All accesses from an individual site are managed by one site handler. This parameter controls the number of site handlers, not the number of individual, simultaneous procedure calls. If you set **number of remote sites** to 5, for example, and each site initiates three remote procedure calls, **sp_who** shows 5 site handler processes for the 15 processes. Only a System Administrator can set **number of remote sites**.

number of remote connections

The **number of remote connections** parameter controls the limit on active remote connections that are initiated to and from this server. This includes those initiated from this server and those from remote sites to this server. Only a System Administrator can set **number of remote connections.**

remote server pre-read packets

All communication between two servers is handled through one site handler to reduce the needed number of connections. This site handler can pre-read and keep track of data packets for each user before the user process that needs them is ready. The **remote server pre-read packets** parameter controls how many packets a site handler will pre-read. The default value 3 is adequate in virtually all cases; higher values can use too much memory. Only a System Administrator can set **remote server pre-read packets**. For more information about configuration parameters, see Chapter 11, “Setting Configuration Parameters” in the *System Administration Guide*.

8 Auditing

Introduction

Auditing is an important part of security in a database management system. The auditing system records security-relevant system activity in an audit trail, which a System Security Officer can use to detect penetration of the system and misuse of resources. A System Security Officer can inspect patterns of access to objects in databases and monitor the activity of specific users. Audit records are traceable to specific users, enabling the audit system to act as a deterrent to users attempting to misuse the system.

In the sections that follow, the audit system is discussed in detail, including what is audited, how the system works, and how to interpret the audit trail.

The Audit System

The audit system consists of:

- The *sybsecurity* database
- System procedures that set the various auditing options
- The audit queue, to which audit records are sent before they are written to the audit trail

The *sybsecurity* Database

The *sybsecurity* database is essential to auditing in SQL Server. It is created as part of the auditing installation process. It contains all system tables found in the *model* database and two additional system tables:

- *sysaudits* – the audit trail
- *sysauditoptions* – contains the global auditing choices

The contents of the audit trail is discussed in detail in "The Audit Trail: sysaudits" on page 8-15.

The Audit System Procedures

Auditing is managed with the following system procedures:

Table 8-1: System procedures used to manage auditing options

System Procedure	Description
sp_auditoption	Enables and disables system-wide auditing and global audit options.
sp_auditdatabase	Establishes auditing of different types of events within a database, or of references to objects within that database from another database.
sp_auditobject	Establishes selective auditing of accesses to tables and views.
sp_auditsproc	Audits the execution of stored procedures and triggers.
sp_auditlogin	Audits a user's attempts to access tables and views, or the text of commands that the user executes.
sp_addauditrecord	Allows users to enter user-defined audit records (comments) into the audit trail.

The Audit Queue

When an audited event occurs, an audit record first goes to the in-memory audit queue, where it is held until it can be processed by the audit process and added to the audit trail. If the audit queue is full when an auditable event occurs, the audit process sleeps until space in the queue becomes available. If this happens repeatedly, performance is degraded. To maximize performance, use a large audit queue. However, large audit queues increase the likelihood of losing audit records if the system crashes.

Therefore, before setting the size of the audit queue, consider the tradeoff between performance and the risk of losing audit records:

Table 8-2: Tradeoff between risk and performance

Queue Size	Performance	Risk of Losing Audit Records
Large	Better	Higher
Small	Worse	Lower

You can configure the size of the audit queue with the **audit queue size** parameter of **sp_configure**. See Chapter 11, "Setting Configuration Parameters," in the *System Administration Guide* for information about how to configure the size of the audit queue and the effects of different queue sizes.

Installing the Audit System

The audit system is installed with **sybinit**, the Sybase installation program. Installation and **sybinit** are discussed in the SQL Server installation and configuration guide for your platform.

sybinit installs the *sybsecurity* database on its own device and the *sysaudits* table on its own segment on that device. This allows you to use the threshold manager to monitor the available space in *sysaudits*. See "Archiving Audit Data" on page 8-18 for more information.

◆ ***WARNING!***

Do not put any objects into the *sybsecurity* database besides those that are automatically installed there. Because the *sysaudits* table is so dynamic, you must carefully monitor the space on its device. This is harder to do if there are extra objects on the device.

Because it is assumed that there will be no extra activity in the *sybsecurity* database, the **trunc log on chkpt** database option is turned on automatically when *sybsecurity* is installed. This means that the log in that database is truncated every time the **checkpoint** process occurs, and the transaction log cannot be dumped or recovered. Inserts into the *sysaudits* table are not logged.

Removing the Audit System

If you need to remove the audit system and reclaim the disk space, follow this procedure:

1. Disable any current auditing with this command:

   ```
   sp_auditoption "enable auditing", "off"
   ```

2. Remove the audit database with this command:

   ```
   drop database sybsecurity
   ```

You cannot remove *sybsecurity* if auditing is enabled. Only a System Security Officer can drop *sybsecurity*.

If you wish to reinstall the audit system at a later time, run **sybinit** again and access the menus that install auditing.

Establishing Auditing

The System Security Officer manages the audit system. Only a user who has been granted that role can:

- Execute any of the auditing system procedures (with the exception of **sp_addauditrecord**)
- Read the audit trail
- Access the audit database

The System Security Officer who is going to manage the audit system must also be granted access to the *sybsecurity* database.

Turning Auditing On and Off

Once you have installed the *sybsecurity* database and the auditing system procedures, you can set the audit options. No auditing actually occurs, however, until you turn auditing on for the server as a whole using this command:

```
sp_auditoption "enable auditing", "on"
```

Turn server-wide auditing off with this command:

```
sp_auditoption "enable auditing", "off"
```

Turning auditing off does not modify your auditing set-up.

sp_auditoption is discussed in the following section.

Setting the Global Audit Options: *sp_auditoption*

The global audit options are those that affect the server as a whole. They are enabled and disabled with **sp_auditoption**, and information about their current settings is contained in the *sybsecurity..sysauditoptions* system table.

The available options and their syntax are described in Table 8-3.

Table 8-3: Global auditing options

Option	Action
enable auditing	Enables or disables system-wide auditing. A System Security Officer must set the **enable auditing** option to **on** before any auditing can take place. Enabling or disabling auditing automatically generates an audit record, so that you can bracket time periods when auditing was enabled. Syntax: `sp_auditoption "enable auditing" [, {"on"|"off"}]`
all	Enables or disables all options **except enable auditing**. **enable auditing** must be set separately. For options that allow selective auditing for successful and/or failed executions, setting **all** to **on** is equivalent to setting all options to **on** or **both**, depending on the option. Syntax: `sp_auditoption "all" [, {"on"|"off"}]`
logins	Enables or disables auditing of successful (**ok**), failed (**fail**), or all (**both**) login attempts by all users. To audit individual users after they have logged on, use **sp_auditlogin**. Syntax: `sp_auditoption "logins" [, {"ok"|"fail"|"both"|"off"}]`
logouts	Enables or disables auditing of all logouts from the server, including unintentional logouts such as dropped connections. Syntax: `sp_auditoption "logouts" [, {"on"|"off"}]`
server boots	Enables or disables generation of an audit record when the server is rebooted. Syntax: `sp_auditoption "server boots" [, {"on"|"off"}]`
rpc connections	When this option is **on**, it generates an audit record whenever a user from another host connects to the local server to run a procedure via a remote procedure call (RPC). Auditing can be enabled for all connection attempts (**both**), successful attempts only (**ok**), or failed attempts only (**fail**). Syntax: `sp_auditoption "rpc connections" [, {"ok"|"fail"|` `"both"|off"}]`
roles	Audits the use of the **set role** command to turn roles on and off. You can enable auditing of all attempts (**both**), successful attempts only (**ok**), or failed attempts only (**fail**). See Chapter 5, "Roles in SQL Server," for more information. Syntax: `sp_auditoption "roles" [, {"ok"|"fail"|"both"|"off"}]`
{**sa** | **sso** | **oper**} **commands**	Audits the use of privileged commands—those requiring one of the roles for execution. You can enable auditing for successful executions only, failed attempts (where failure is due to the user lacking the proper role), or both. See "Roles" in the *SQL Server Reference Manual* for a list of the commands that require the various roles. Syntax: `sp_auditoption "{sa|sso|oper}` `commands" [, {"ok"|"fail"|"both"|"off"}]`

Table 8-3: Global auditing options (continued)

Option	Action
errors	Audits fatal errors (errors that break the user's connection to the server and require the client program to be restarted), nonfatal errors, or both. Fatal errors do not include server internal fatal software errors (bus errors, segmentation faults, etc.) In case of internal errors, information will be contained in the error log file for the server. Syntax: `sp_auditoption "errors" [, {"nonfatal"|"fatal" |"both"|"off"}]`
adhoc records	Allows users to send text to the audit trail via the **sp_addauditrecord** command. Syntax: `sp_auditoption "adhoc records", {"on"|"off"}`

The initial, default value of all options is "off."

Examples

Using **sp_auditoption** with no parameters, like this:

```
sp_auditoption
```

or with the **all** option, like this:

```
sp_auditoption "all"
```

displays the current settings for all of the global audit options.

Using **sp_auditoption** with the name of any option and no other parameters displays the current setting for that option. For example:

```
sp_auditoption "logins"
```

displays the current status of the **logins** option.

This command:

```
sp_auditoption "errors", "fatal"
```

establishes auditing of fatal errors.

This command:

```
sp_auditoption "logins", "both"
```

establishes auditing of all attempts to log into SQL Server, both successful and unsuccessful. To audit individual users, use **sp_auditlogin**.

Auditing Users: *sp_auditlogin*

sp_auditlogin allows you to audit a SQL Server user's attempts to access tables and views in any database, and the text of commands that the user sends to SQL Server.

Using **sp_auditlogin** with no parameters, like this:

```
sp_auditlogin
```

displays the audit status for all login names in the server.

Using **sp_auditlogin** with a login name and no other parameters, like this:

```
sp_auditlogin bob
```

displays the audit status for that user.

Auditing a User's Access to Tables and Views

The syntax for auditing a user's attempts to access tables and views is:

```
sp_auditlogin [login_name [, "table" | "view"
   [, "ok"|"fail"|"both"|"off"]]]
```

An **access** is the use of the **select, insert, update,** or **delete** command on a table or view. The **table** option audits *login_name*'s attempts to access tables in any database. **view** audits *login_name*'s attempts to access views in any database. If you use **table** or **view** without the **ok** | **fail** | **both** | **off** parameter, **sp_auditlogin** displays the status of table or view auditing for the named user.

The **ok** parameter enables auditing for successful table or view accesses only. **fail** establishes auditing of accesses that fail because the user lacks permissions on an object. **both** establishes auditing of both successful and failed accesses. **off** disables auditing of the named type (**table** or **view**).

For example, this command:

```
sp_auditlogin "joe", "table", "both"
```

audits all of Joe's attempts to access any table on the server.

This command:

```
sp_auditlogin joe, "table", "fail"
```

audits Joe's attempts to access tables on which he has not been granted permission.

This command:

```
sp_auditlogin bob, "table"
```

displays the status of table auditing for Joe.

Auditing the Text of a User's Commands

The syntax for auditing the text of commands that a user sends to SQL Server is:

```
sp_auditlogin [login_name [, "cmdtext"
   [, "on"|"off"]]]
```

Set **cmdtext on** to write the text of a user's SQL Server commands to the audit trail.

Using the **cmdtext** option without the final parameter, like this:

```
sp_auditlogin bob, "cmdtext"
```

displays the status of **cmdtext** auditing for Bob.

Auditing Databases: *sp_auditdatabase*

The **sp_auditdatabase** system procedure permits you to establish selective auditing on databases. The following are the auditable events that can occur in or on a database:

- Use of the **drop**, **grant**, **revoke**, and **truncate table** commands within a database
- Use of the **drop database** and **use** commands on a database
- Execution of SQL commands from within another database that reference the audited database

sp_auditdatabase Syntax

The syntax of **sp_auditdatabase** is:

```
sp_auditdatabase [dbname [, {"ok"|"fail"|"both"|"off"}
   [, "d u g r t o"}]]]
```

- If you use **sp_auditdatabase** with no parameters, it displays the current audit status for all databases in the server.
- *dbname* is the name of the database you want to audit. If you use **sp_auditdatabase** with *dbname* and no other arguments, it displays the audit settings for the specified database.
- **ok** establishes auditing of successful executions of the commands specified in the third parameter. **fail** audits access attempts that

fail because the user lacks permission to access the database. **both** audits successful and failed executions. **off** turns off the specified type of auditing for the named database.

- **d**, **u**, **g**, **r**, **t**, and **o** are the types of database events that you can audit. They are described in the following table:

Table 8-4: Database event types

Event Type	Meaning
d	Audits execution of the **drop table**, **drop view**, **drop procedure**, or **drop trigger** commands within *dbname*, and execution of the **drop database** command when *dbname* is being dropped.
u	Audits execution of the **use** command on *dbname*.
g	Audits execution of the **grant** command within *dbname*.
r	Audits execution of the **revoke** command within *dbname*.
t	Audits execution of the **truncate table** command within *dbname*.
o	"Outside access"; audits execution of SQL commands from within another database that refer to objects in *dbname*.

Choose one or more event types, in any order. If none are specified, the **ok** | **fail** | **both** | **off** argument applies to all event types.

For example, this command:

```
sp_auditdatabase pubs2, "ok", "du"
```

enables auditing of successful executions of the **drop** and **use** commands on *pubs2*.

You can execute **sp_auditdatabase** more than once on a database, and the options that you set will accumulate with each execution. Therefore, you can set some options for success only, others for failure only, and still others for both success and failure. For example, this sequence:

```
sp_auditdatabase pubs2, "ok", "d"
go
sp_auditdatabase pubs2, "fail", "u"
go
sp_auditdatabase pubs2, "both", "gr"
go
```

establishes auditing on *pubs2* of successful **drop** commands, failed **use** commands, and all **grant** and **revoke** commands.

If, later, you no longer want to audit the execution of the **drop** command, issue this statement:

```
sp_auditdatabase pubs2, "off", "d"
```

If you wish to turn off all auditing of *pubs2*, use this command:

```
sp_auditdatabase pubs2, "off"
```

Auditing Tables and Views: *sp_auditobject*

sp_auditobject allows you to audit accesses of specified tables and views. An **access** is the use of the **select, insert, update,** or **delete** command on a table or view. You can establish auditing of specified tables and views, or create default audit settings for newly-created tables and views.

To establish auditing of an existing table or view, the syntax is:

```
sp_auditobject [owner.]objname, dbname
   [, "ok"|"fail"|"both"|"off" [,"{d u s i}"]]
```

To establish default audit settings for future tables and views in the current database, the syntax is:

```
sp_auditobject {"default table"|"default view"},
   dbname [, "ok"|"fail"|"both"|"off"
   [, "{d u s i}"]]
```

Notes on the keywords and parameters are as follows:

- If you are establishing auditing for an existing table or view, use the *objname* parameter. *objname* is the name of an existing table or view in the current database. If you are not the owner of the object, or the object is not owned by the Database Owner ("dbo"), you must qualify the object name with the name of the owner, like this:

  ```
  "ownername.objname"
  ```

- If you are establishing default audit settings for future tables or views in the current database, use the **default table** | **default view** parameter. This specifies that these audit settings are to be the defaults for any tables and views that will be created in the future. These default settings are not applied to any tables or views that exist at the time that **sp_auditobject** is executed. If you do not set any default audit settings in a database, newly-created tables and views will have no auditing options set initially.

- If you are using the *objname* parameter, *dbname* must be the name of the current database. You must be in that database to execute **sp_auditobject**, and you must also supply its name.

  If you are using the **default table** | **default view** option, *dbname* can be the name of any database.

- **ok** establishes auditing for accesses that are successful. **fail** audits attempts to access the object that fail because the user has not been granted permission on the object. **both** audits both types of access attempts. **off** turns auditing off for the named table or view.

- The **d u s i** parameter indicates what kinds of accesses to audit. Choose one or more, in any order. If none are specified, the **ok** | **fail** | **both** | **off** argument applies to all four event types. The types of access are:

Table 8-5: Types of object access

Parameter	Meaning
d	delete
u	update
s	select
i	insert

Examples

This command:

```
sp_auditobject authors, pubs2, "both", "dui"
```

establishes auditing on the *authors* table in the *pubs2* database. Both successful and failed attempts to execute the **delete**, **update**, and **insert** commands on *authors* will be audited.

You can execute **sp_auditobject** more than once on a table or view, and the options that you set will accumulate with each execution. Therefore, you can set some options for success only, others for failure only, and still others for both success and failure. For example, this sequence:

```
sp_auditobject titles, pubs2, "ok", "id"
go
sp_auditobject titles, pubs2, "fail", "u"
go
```

establishes auditing on the *titles* table so that successful executions of the **insert** and **delete** commands will be audited, and failed attempts to execute **update** commands will be audited.

This command:

```
sp_auditobject titles, pubs2, "off"
```

turns off all auditing on the *titles* table.

This command:

```
sp_auditobject publishers
```

displays the current audit settings for the *publishers* table.

This command:

```
sp_auditobject "default table", pubs2, "fail", "du"
```

establishes auditing of failed attempts to execute the **delete** and **update** commands on all new tables in the *pubs2* database. Any table created after this command is executed will automatically have these audit settings.

This command:

```
sp_auditobject "default view", pubs2
```

displays the default audit settings for views in the *pubs2* database.

Auditing Stored Procedures: *sp_auditsproc*

The **sp_auditsproc** system procedure allows you to audit the execution of stored procedures and triggers. The values of any parameters passed to a stored procedure are also audited. You can establish auditing of existing stored procedures or triggers, or create default audit settings for future stored procedures and triggers.

The syntax to establish auditing of existing stored procedures and triggers is:

```
sp_auditsproc {sproc_name | "all"},
   dbname [, {"ok"|"fail"|"both"|"off"}]
```

The syntax to establish default audit settings for future stored procedures and triggers in the current database is:

```
sp_auditsproc "default", dbname
   [, {"ok"|"fail"|"both"|"off"}]
```

Notes on the keywords and parameters are as follows:

- If you are establishing auditing for an existing stored procedure or trigger, use the *sproc_name* parameter. *sproc_name* is the name

of the stored procedure or trigger on which you want to establish auditing. If you are not the owner of the object, or you are not the Database Owner ("dbo"), you must qualify the name of the stored procedure or trigger with the owner's name, like this:

```
"ownername.sproc_name"
```

If you wish to establish auditing for all existing triggers and stored procedures in the current database, specify **all**.

- If you are establishing default audit settings for future stored procedures and triggers in the current database, use the **default** parameter. This specifies that these audit settings are to be the defaults for any stored procedures and triggers that will be created in the future. These default settings are not applied to any stored procedures or triggers that exist at the time that **sp_auditsproc** is executed. If you do not set any default audit settings in a database, newly-created stored procedures and triggers will not be audited.
- If you are using the *sproc_name* | **all** parameter, *dbname* must be the name of the current database. You must be in this database to execute **sp_auditsproc**, and you must also supply its name.

  If you are using the **default** parameter, *dbname* can be the name of any database.
- **ok** establishes auditing for successful executions of the named stored procedure or trigger. **fail** audits attempts to execute the procedure that fail because the user has not been granted permission on the stored procedure, or because the user does not have permission on the underlying table or view. (Permissions are not granted on triggers, so this option does not apply to them.) **both** audits both successful and failed execution attempts. **off** turns auditing off for the named stored procedure or trigger.

Examples

When used with no parameters, like this:

```
sp_auditsproc
```

sp_auditsproc displays the auditing status of all stored procedures and triggers in the current database.

When used with the name of a stored procedure or trigger and no other parameters, like this:

```
sp_auditsproc new_sproc
```

sp_auditsproc displays the audit status for *new_sproc*.

When used with the **default** parameter and no others, like this:

```
sp_auditsproc default
```

sp_auditsproc displays the default audit settings for stored procedures and triggers in the current database.

This command:

```
sp_auditsproc "all", pubs2, "fail"
```

establishes auditing of failed executions of all stored procedures and triggers in the *pubs2* database.

This command:

```
sp_auditsproc "default", pubs2, "ok"
```

sets a default in the *pubs2* database so that successful executions of new stored procedures and triggers will be audited.

Adding User-Specified Records to the Audit Trail

sp_addauditrecord allows users to enter user-defined audit records (comments) into the audit trail. The syntax is:

```
sp_addauditrecord [text] [, db_name] [, obj_name]
   [, owner_name] [, dbid] [, objid]
```

All of the parameters are optional.

- *text* – Text of the message that you wish to add to *sysaudits*. The text is inserted into the *extrainfo* field of *sysaudits*.
- *db_name* – Name of the database referred to in the record. This is inserted into the *dbname* field of *sysaudits*.
- *obj_name* – Name of the object referred to in the record. This is inserted into the *objname* field of *sysaudits*.
- *owner_name* – Owner of the object referred to in the record. This is inserted into the *objowner* field of *sysaudits*.
- *dbid* – Database ID number of *db_name*. This is an integer value, and must not be placed in quotes. *dbid* is inserted into the *dbid* field of *sysaudits*.
- *objid* – Object ID number of *obj_name*. This is an integer value and must not be placed in quotes. *objid* is inserted into the *objid* field of *sysaudits*.

You can use **sp_addauditrecord** if:

- You have execute permission on **sp_addauditrecord.** (No special role is required.)
- Auditing is enabled (**sp_auditoption "enable auditing"** is set to **on**).
- The **adhoc records** option of **sp_auditoption** is set to **on.**

sp_addauditrecord does not check the correctness of the information you enter. For example, it does not check to see if the database ID you have entered is correct for the database name that you specify.

When the audit system is installed, only a System Security Officer and the Database Owner of *sybsecurity* can use **sp_addauditrecord.** Permission to execute it may be granted to other users.

Examples

The following example adds a record to *sysaudits.* The message portion is entered into the *extrainfo* field of *sysaudits*, "corporate" into the *dbname* field, "payroll" into the *objname* field, "dbo" into the *objowner* field, "10" into the *dbid* field, and "1004738270" into the *objid* field:

```
sp_addauditrecord "I gave A. Smith permission to
view the payroll table in the corporate database.
This permission was in effect from 3:10 to 3:30 pm
on 9/22/92.", "corporate", "payroll", "dbo", 10,
1004738270
```

The following example inserts information only into the *extrainfo* and *dbname* fields of *sysaudits*:

```
sp_addauditrecord @text="I am disabling auditing
briefly while we reconfigure the system",
@db_name="corporate"
```

The Audit Trail: *sysaudits*

sysaudits is a special system table that exists only in the *sybsecurity* database and contains the audit trail. The only operations allowed on it are the **select** and **truncate** commands, and these may be executed only by a System Security Officer.

Following are the columns of *sysaudits*:

- *event* – Contains a number corresponding to the event that is being audited. Table 8-6 describes the events and the contents of the *extrainfo* column that correspond to each event.

Table 8-6: Contents and description of event and extrainfo columns

event No.	Description	Contents of *extrainfo* Column
1	Enable auditing	NULL
2	Disable auditing	NULL
3	Login	Host name
4	Logout	Host name
5	Server boot	Names of the server program, master device, interfaces file path, server, and error log file
6	RPC connection	Remote server name, host name
7	Use of **set** command to turn roles on and off	Role, new setting
8	Command requiring **sa_role** role	Command type
9	Command requiring **sso_role** role	Command type
10	Command requiring **oper_role** role	Command type
12	Command requiring **navigator** role	Command type
13	Error	Error number, severity, and state
14	Ad hoc audit record	User-supplied comment text
15	Command requiring **replication_role**	Command type
100	Database reference	Command type
101	Table reference	Command type
102	View reference	Command type
103	Stored procedure execution	Parameter list
104	Trigger execution	NULL
105	User's attempts to access a table	Command type
106	User's attempts to access a view	Command type
107	User's command text auditing	Command batch text

- *eventmod* (modifier) – Supplements the *event* field by further categorizing the type of event being audited. For example, this field will contain a different value for a successful login than for a failed login.

Possible values for the *eventmod* field are:

Table 8-7: Values for the eventmod field in sysaudits

Value in *eventmod* Field	Description
0	No *eventmod* for this audited event.
1	Successful occurrence of this audited event. For error auditing (*event* code 12), a nonfatal error.
2	Failed occurrence of this audited event. For error auditing (*event* code 12), a fatal error.

- *spid* (server process ID) – ID number of the server process that caused the audit record to be written.
- *eventtime* – Date and time that the audited event occurred.
- *sequence* – Sequence number of the record within a single event (some events generate more than one audit record).
- *suid* – Server user ID of the user who performed the audited function.
- *dbid* – Depending on the type of event being audited, this is the ID of the database in which:
  - The object, stored procedure, or trigger resides, or
  - The audited event occurred.
- *objid* – ID of the object accessed by the audited event. When a stored procedure or trigger is being audited, this field contains the ID of the relevant stored procedure or trigger.
- *xactid* – Transaction ID of the transaction containing the audited event. This can be useful for associating the audit record to a specific transaction in the transaction log. For a multi-database transaction, this field contains the transaction ID from the database where the transaction originated.
- *loginname* – Login name of the user who performed the audited function.
- *dbname* – Name of the database corresponding to *dbid*.
- *objname* – Name of the object corresponding to *objid*.
- *objowner* – Name of the owner of the object accessed by the audited event.

- *extrainfo* – Additional information regarding the audited event. See Table 8-6 on page 8-16 for a description of the contents of *extrainfo* for each event.

Reading the Audit Trail

The *sysaudits* table can be accessed only by a System Security Officer, who can read the audit record by executing SQL commands on it. The only commands that are allowed on *sysaudits* are **select** and **truncate**.

For example, the following command requests audit records for tasks performed by "bob" on July 5, 1993:

```
select * from sysaudits
where loginname = "bob"
and eventtime like "Jul 5% 93"
```

The following command requests audit records for commands performed on or in the *pubs2* database by users with the System Security Officer role (that kind of role auditing is *event* type 9, as shown in Table 8-6 on page 8-16):

```
select * from sysaudits
where event = 9
and dbname = "pubs2"
```

Archiving Audit Data

Because audit data is continuously added to the *sysaudits* table, it is necessary to archive old audit data from time to time, depending on the size of the device on which *sysaudits* resides. If the audit device fills up, audited system activity comes to a halt.

To archive audit data, you can:

- Use the **select into** command to create a new table and copy data into it.
- Use the **insert** and **select** commands to copy data from *sysaudits* into a pre-existing table.

Put the archive table in a separate database on a separate device from the *sybsecurity* database. It is preferable that this database be a special archive database, where you can preserve the audit records for as long as you need to.

The following sections describe the archive procedures.

Using *select into*

The **select into** command creates a new table based on the table from which you are selecting. Before you can use **select into**, you must turn on the **select into/bulkcopy** database option in the archive database (with **sp_dboption**). Use the following procedure to copy *sysaudits*:

1. Create the archive database on a separate device from the one containing *sysaudits*.
2. In that database, use **sp_dboption** to turn on the **select into/bulk copy** option.
3. Copy the contents of *sysaudits* into this database with a command like this:

   ```
   select * into auditarchive7_5
   from sybsecurity.dbo.sysaudits
   ```

   You can use the **where** clause to further qualify the rows to be selected from *sysaudits* if you wish to.
4. Return to the *sybsecurity* database and delete all rows from *sysaudits* with these commands:

   ```
   use sybsecurity
   go
   truncate table sysaudits
   ```

While the **select into/bulkcopy** option is on, you are not allowed to dump the transaction log, because these operations are not logged and changes are therefore not recoverable from transaction logs. In this situation, issuing the **dump transaction** statement produces an error message instructing you to use **dump database** instead.

Be certain that you dump your database before you turn off the **select into/bulkcopy** option.

See **select** in the *SQL Server Reference Manual* for more information.

Using *insert* and *select* to Copy Into an Archive Table

You can use this method to copy the audit data into an existing table that has the same columns as *sysaudits*:

1. Create the archive database on a separate device from the one containing *sysaudits*.
2. Create an archive table with columns identical to those in *sysaudits*. If such a table does not already exist, you can use **select**

into to create an empty one by using a false condition in the **where** clause. For example:

```
select *
into auditarchive7_5
from sybsecurity.dbo.sysaudits
where 1 = 2
```

The **where** condition is always false, so an empty table is created that is a duplicate of *sysaudits*.

The **select into/bulk copy** database option must be turned on in the archive database (using **sp_dboption**) before you can use **select into**.

3. Insert all rows from *sysaudits* into the archive table, like this:

```
insert auditarchive7_5
select * from sybsecurity.dbo.sysaudits
```

4. Delete all rows from *sysaudits* with this command:

```
truncate table sysaudits
```

Using a Threshold Action Procedure

sysaudits is automatically installed on its own segment on the audit device. Therefore, you can use the **sp_addthreshold** system procedure to specify a user-supplied "threshold action" procedure to be executed when the free space left on the *sysaudits* segment goes below a specified threshold. (Threshold management is discussed in Chapter 21, "Managing Free Space with Thresholds," in the *System Administration Guide*.)

The "threshold action" stored procedure should perform at least these two tasks:

1. Execute a **select** or **insert** with **select** statement from *sysaudits* to an archive table, preferably in a separate archive database.
2. Truncate the *sysaudits* table with the **truncate table** command.

You can then manage the archived audit data with **dump** and **load** commands as desired.

If *sysaudits* Becomes Full

If you do not regularly archive your audit data, the *sysaudits* table may run out of space. This, in turn, causes the audit device to become full. The following sections describe:

- What happens when the *sysaudits* table is full
- How to recover

What Happens If *sysaudits* Becomes Full

If *sysaudits* is full, the following occurs:

1. The audit process attempts to insert the next audit record into *sysaudits*. This fails, so the audit process terminates. An error message goes to the error log.
2. When a user attempts to perform an auditable event (an event on which auditing has been enabled), the event cannot be completed because auditing cannot proceed. The user process terminates. Users who do not attempt to perform an auditable event are unaffected.
3. If you have global login auditing enabled (with **sp_auditoption "logins"**), no one can log into the server, including a System Security Officer.
4. If you are auditing commands requiring System Security Officer privileges (with **sp_auditoption "sso_role", "on"**), a System Security Officer will be unable to execute any SSO-specific commands, such as turning auditing off.

Recovering When *sysaudits* Is Full

If the *sysaudits* device becomes full or the audit process terminates abnormally for any reason, the System Security Officer immediately obtains these special privileges:

- The System Security Officer becomes exempt from auditing. Every auditable event performed by a System Security Officer after this point causes a warning message to be sent to the error log file. The message states the date and time and a warning that an audit has been missed, as well as the user's server user ID, login name, *event* code, and any information that would go into the *extrainfo* field.
- The System Security Officer can execute the **shutdown** command to shut down the server after *sysaudits* has been archived and truncated. Normally, only the System Administrator can execute **shutdown**.

If *sysaudits* is full, the System Security Officer can now archive and truncate *sysaudits* as described in "Archiving Audit Data" on page

8-18. He or she can then execute **shutdown** to stop the server. A System Administrator can then restart the server, which will reestablish auditing.

If the audit process fails for a reason other than *sysaudits* becoming full, please call Sybase Technical Support.

Glossary

automatic recovery

A process that runs every time SQL Server is stopped and restarted. The process ensures that all transactions that have completed before the server went down are brought forward and all incomplete transactions are rolled back.

backup

A copy of a database or transaction log, used to recover from a media failure.

batch

One or more Transact-SQL statements terminated by an end-of-batch signal, which submits them to SQL Server for processing. The Report Workbench™ and other client software supply end-of-batch signals to SQL batches automatically.

built-in functions

A wide variety of functions that take one or more parameters and return results. The built-in functions include mathematical functions, system functions, string functions, text functions, date functions, and a type conversion function.

bulk copy

The utility for copying data in and out of databases, called **bcp**.

character set

A set of specific (usually standardized) characters with an encoding scheme that uniquely defines each character. ASCII and ISO 8859-1 (Latin 1) are two common character sets.

character set conversion

Changing the encoding scheme of a set of characters on the way into or out of SQL Server. Conversion is used when SQL Server and a client communicating with it use different character sets. For example, if SQL Server uses ISO 8859-1 and a client uses Code Page 850, character set conversion must be turned on so that both server and client interpret the data passing back and forth in the same way.

checkpoint

The point at which all data pages that have been changed are guaranteed to have been written to the database device.

clustered index

An index in which the physical order and the logical (indexed) order is the same. The leaf level of a clustered index represents the data pages themselves.

code set

See **character set**.

collating sequence

See **sort order**.

command

An instruction that specifies an operation to be performed by the computer. Each command or SQL statement begins with a keyword, such as **insert**, that names the basic operation performed. Many SQL commands have one or more keyword phrases, or clauses, that tailor the command to meet a particular need.

command terminator

A command terminator is the end-of-batch signal that sends the batch to SQL Server for processing.

context-sensitive protection

Context-sensitive protection provides certain permissions or privileges, depending on the identity of the user. This type of protection can be provided by SQL Server using a view and the **user_id** built-in function.

conversion

See **character set conversion**.

data definition

The process of setting up databases and creating database objects such as tables, indexes, rules, defaults, constraints, procedures, triggers, and views.

data dictionary

The system tables that contain descriptions of the **database objects** and how they are structured.

data modification

Adding, deleting, or changing information in the database with the **insert**, **delete**, and **update** commands.

database

A set of related data tables and other database objects that are organized and presented to serve a specific purpose.

database device

A device dedicated to the storage of the objects that make up databases. It can be any piece of disk or a file in the file system that is used to store databases and database objects.

database object

A database object is one of the components of a database: table, view, index, procedure, trigger, column, default, constraint, or rule.

Database Owner

The user who creates a database becomes the Database Owner. A Database Owner has control over all the database objects in that database. The login name for the Database Owner is "dbo".

datatype

Specifies what kind of information each column will hold, and how the data will be stored. Datatypes include *char*, *int*, *money*, and so on. Users can construct their own datatypes in SQL Server based on the SQL Server system datatypes. User-defined datatypes are not supported in Sybase SQL Toolset™.

date function

A function that displays information about dates and times, or manipulates date or time values. The five date functions are **getdate, datename, datepart, datediff** and **dateadd.**

dbo

In a user's own database, SQL Server recognizes the user as "dbo". A database owner logs into SQL Server using his or her assigned login name and password.

deadlock

A situation which arises when two users, each having a **lock** on one piece of data, attempt to acquire a lock on the other's piece of data. The SQL Server detects deadlocks, and kills one user's process.

default

The option chosen by the system when no other option is specified.

default database

The database that a user gets by default when he or she logs in.

default language

1. The default language of a user is the language that displays that user's prompts and messages. It can be set with **sp_modifylogin** or the **language** option of the **set** command.

2. The SQL Server default language is the language that is used to display prompts and messages for all users unless a user chooses a different language.

demand lock

A demand lock prevents any more shared locks from being set on a data resource (table or data page). Any new shared lock request has to wait for the demand lock request to finish.

dirty read

"Dirty reads" occur when one transaction modifies a row, and then a second transaction reads that row before the first transaction commits the change. If the first transaction rolls back the change, the information read by the second transaction becomes invalid.

discretionary access controls (DAC)

Restricts access to objects based on identity and/or group membership. The controls are discretionary in the sense that a user with a certain access permission (for example, an object owner) is capable of passing access permission on to any other user (such as with the **grant** command).

disk allocation pieces

Disk allocation pieces are the groups of allocation units from which SQL Server constructs a new database file. The minimum size for a disk allocation piece is one allocation unit, or 256 2K pages (256 4K pages on Stratus).

disk initialization

The process of preparing a database partition, foreign device or file for SQL Server use. Once the device is initialized, it can be used for storing databases and database objects. The command used to initialize a database device is **disk init**.

disk mirror

A duplicate of a SQL Server **database device**. All writes to the device being mirrored are copied to a separate physical device, making the second device an exact copy of the device being mirrored. If one of the devices fails, the other contains an up-to-date copy of all transactions. The command **disk mirror** starts the disk mirroring process.

display precision

The number of significant binary digits offered by the default display format for real and float values. Internally, real and float values are stored with a precision less than or equal to that of the platform-specific datatypes on which they are built. For display purposes, Sybase *real* values have 9 digits of precision; Sybase *float* values, 17.

dump striping

Interleaving of dump data across several **dump volumes**.

dump volume

A single tape, partition, or file used for a database or transaction dump. A dump can span many volumes, or many dumps can be made to a single tape volume.

dynamic dump

A dump made while the database is active.

engine

A process running a SQL Server that communicates with other server processes using shared memory. An engine can be thought of as one CPU's worth of processing power. It does not represent a particular CPU on a machine. Also referred to as "server engine."A SQL Server running on a uniprocessor machine will always have one engine, engine 0. A SQL Server running on a multiprocessor machine can have one or more engines. The maximum number of engines running on SQL Server can be reconfigured using the **max online engines** configuration variable.

error message

A message that SQL Server issues, usually to the user's terminal, when it detects an error condition.

error state number

The number attached to a SQL Server error message that allows unique identification of the line of SQL Server code at which the error was raised.

exclusive locks

Locks that prevent any other transaction from acquiring a lock until the original lock is released at the end of a transaction, always applied for update (**insert**, **update**, **delete**) operations.

expression

Returns values. An expression can be a computation, column data, a built-in function, or a subquery.

format file

A file created while using **bcp** to copy data out from a table in a SQL Server database to an operating system file. The format file contains information on how the data being copied out is formatted and can be used to copy the data back into a SQL Server table or to perform additional copy outs.

free-space threshold

A user-specified threshold that specifies the amount of space on a segment, and the action to be taken when the amount of space available on that segment is less than the specified space.

functions

See **built-in functions**.

global variable

System-defined variables that SQL Server updates on an ongoing basis. For example, *@@error* contains the last error number generated by the system.

guest

If the user name "guest" exists in the *sysusers* table of a database, any user with a valid SQL Server login can use that database, with limited privileges.

hexadecimal string

A hexadecimal-encoded binary string that begins with the prefix 0x and can include the digits 0 through 9 and the uppercase and lowercase letters A through F. The interpretation of hexadecimal strings is platform specific. For some systems, the first byte after the prefix is the most significant; for others, the last byte is most significant. For example, the string 0x0100 is interpreted as 1 on some systems and as 256 on others.

identifier

A string of characters used to identify a database object, such as a table name or column name.

initialization

See **disk initialization**.

int

A signed 32-bit integer value.

intent lock

An intent lock indicates the intention to acquire a shared or exclusive lock on a data page.

isolation level

Also called "locking level," isolation level specifies the kinds of actions that are not permitted while the current transaction executes. The ANSI standard defines 3 levels of isolation for SQL transactions. Level 1 prevents **dirty reads**, and level 2 also prevents **non-repeatable reads**. Level 3 prevents both types of reads and **phantoms**; it is equivalent to doing all **select**s with **holdlock**. The user controls the isolation level with the **set** option **transaction isolation level**; the default is isolation level 1.

kernel

A module within SQL Server that acts as the interface between SQL Server and the operating system.

keyword

A word or phrase that is reserved for exclusive use by Transact-SQL. Also called a "reserved word."

last-chance threshold

A default threshold in SQL Server that suspends or kills user processes if the transaction log has run out of room. This threshold leaves just enough space for the deallocation records for the pages cleared by the log dump itself. Threshold events call a user-defined procedure whose default name is **sp_thresholdaction.** This procedure is **not** supplied by Sybase; it must be written by the System Administrator.

leaf level

The bottom level of a clustered or non-clustered index. In a clustered index, the leaf level contains the actual data pages of the table.

livelock

A request for an **exclusive lock** that is repeatedly denied because a series of overlapping **shared locks** keeps interfering. SQL Server detects the situation after four denials and refuses further shared locks.

locking

The process of restricting access to resources in a multi-user environment to maintain security and prevent concurrent access problems. SQL Server automatically applies locks to tables or pages.

locking level

See **isolation level**.

login

The name a user uses to log into SQL Server. A login is valid if SQL Server has an entry for that user in the system table *syslogins*.

mandatory access controls (MAC)

Restricts access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (that is, clearance) of users to access information of such sensitivity. This security feature is available with Secure SQL Server™, but not with the standard SQL Server.

master database

Controls the user databases and the operation of SQL Server as a whole. Known as *master*, it keeps track of such things as user accounts, ongoing processes, and system error messages.

message number

The number that uniquely identifies an error message.

mirror

See **disk mirror**.

model database

A template for new user databases. The **buildmaster** program and the **installmodel** script create *model* when SQL Server is installed. Each time the **create database** command is issued, SQL Server makes a copy of *model* and extends it to the size requested, if necessary.

multibyte character set

A character set that includes characters encoded using more than one byte. EUC JIS and Shift-JIS are examples of character sets that include several types of characters represented by multiple bytes in a Japanese language environment.

non-clustered index

An index that stores key values and pointers to data. The leaf level points to data pages rather than containing the data itself. Compare to **clustered index**.

non-repeatable read

Non-repeatable reads occur when one transaction reads a row and then a second transaction modifies that row. If the second transaction commits its change, subsequent reads by the first transaction yield different results than the original read.

normalization rules

The standard rules of database design in a relational database management system.

null

Having no explicitly assigned value. NULL is not equivalent to zero, or to blank. A value of NULL is not considered to be greater than, less than, or equivalent to any other value, including another value of NULL.

object

A passive entity that contains or receives information, and that cannot change the information it contains. In SQL Server, objects can include rows, tables, databases, stored procedures, and views. See also **database objects**.

object access permissions

Permissions that regulate the use of certain commands (data modification commands plus **select**, **truncate table**, and **execute**) to specific tables, views, columns, or procedures. These permissions are granted and revoked by the owner of the object, who can grant them to other users. See also **object creation permissions**.

object creation permissions

Permissions that regulate the use of commands that create objects (for example, **create table**, **create procedure**, and **create database**). These permissions can be granted only by a System Administrator or a Database Owner. See also **object access permissions**.

operating system

A group of programs that translates your commands to the computer, helping you perform such tasks as creating files, running programs, and printing documents.

parameter

1. An argument to a stored procedure.

2. A value passed between routines and / or forms.

permission

The authority to perform certain actions on certain database objects or to run certain commands.

phantoms

Phantoms occur when one transaction reads a set of rows that satisfy a search condition, and then a second transaction modifies the data (through an **insert**, **delete**, **update**, and so on). If the first transaction repeats the read with the same search conditions, it obtains a different set of rows.

precision

A positive integer that determines the maximum number of digits that can be represented in a decimal, numeric, or float column.

privilege

See **permission**.

query

1. A request for the retrieval of data with a **select** statement.

2. Any SQL statement that manipulates data.

read access

Permission to read an object (for example, to select rows from a table).

recovery

The process of rebuilding one or more databases from database dumps and log dumps. See also **automatic recovery**.

remote procedure calls

A stored procedure executed on a server different from the server the user is logged into.

roles

Privileges granted to identified users to perform various administrative, operational, and security-related tasks. In SQL Server, the available roles are System Administrator, System Security Officer, and Operator.

sa account

The single login, "sa", which is created when SQL Server is installed. This login is configured with both the System Administrator and System Security Officer roles. To increase individual accountability, use the "sa" account to assign roles to individual logins.

scale

A non-negative integer that determines the maximum number of digits that can be represented to the right of the decimal point. The scale of a datatype cannot be greater than its **precision**.

schema

A persistent object in the database. It consists of the collection of objects associated with a particular schema name and user authorization identifier. The objects are tables, views, domains, constraints, assertions, privileges and so on. A schema is created by a **create schema** statement.

Secure SQL Server

A multilevel trusted database management system that is targeted for evaluation at the Class B1 criteria. It is an enhanced version of the standard SQL Server, which is targeted for evaluation at the Class C2 criteria. The requirements for both 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." The Secure SQL Server adds security functions to those offered by the standard server, including **mandatory access controls**. See also **SQL Server** and **mandatory access controls**.

segment

A named subset of database devices available to a particular database. It is a label that points to one or more database devices. Segments can be used to control the placement of tables and indexes on specific database devices.

server engine

See **engine**.

server user ID

The ID number by which a user is known to SQL Server.

severity level number

The severity of an error condition: errors with severity levels of 19 and higher are fatal errors.

shared lock

A lock created by non-update ("read") operations. Other users may read the data concurrently, but no transaction can acquire an **exclusive** lock on the data until all the shared locks have been released.

sort order

Used by SQL Server to determine the order in which character data is sorted. Also called "collating sequence".

SQL Server

The server in Sybase's client-server architecture. SQL Server manages multiple databases and multiple users, keeps track of the actual location of data on disks, maintains mapping of logical data description to physical data storage, and maintains data and procedure caches in memory. SQL Server supports security features such as discretionary access controls and division of roles. The Server is targeted to evaluate at the Class C2 criteria.

SSO

See **System Security Officer**.

statement

A statement begins with a **keyword** that names the basic operation or command to be performed.

stored procedure

A collection of SQL statements and optional control-of-flow statements stored under a name. SQL Server-supplied stored procedures are called **system procedures**.

subject

An active entity that can manipulate database objects. In SQL Server, subjects include users and processes acting on behalf of users.

System Administrator

A user in charge of SQL Server system administration, including managing disk storage, granting and revoking the System Administrator role, and creating new databases. See also **sa account**.

system databases

The four databases on a newly installed SQL Server: the *master* database, which controls user databases and the operation of SQL Server; the temporary database (*tempdb*), used for temporary tables; the system procedures database (*sybsystemprocs*), and the model database (*model*), which is used as a template to create new user databases.

system function

A function that returns special information from the database, particularly from the system tables.

system procedures

Stored procedures that SQL Server supplies for use in system administration. These procedures are provided as shortcuts for retrieving information from the system tables, or mechanisms for accomplishing database administration and other tasks that involve updating system tables.

System Security Officer

A user who controls security-related operations in SQL Server, including auditing, locking and unlocking login accounts, and password management. See also **sa account**.

system table

One of the data dictionary tables. The system tables keep track of information about SQL Server as a whole and about each user database. The *master* database contains some system tables that are not in user databases.

temporary database

The temporary database (*tempdb*) in SQL Server that provides a storage area for temporary tables and other temporary working storage needs (for example, intermediate results of **group by** and **order by**).

transaction

A mechanism for ensuring that a set of actions is treated as a single unit of work.

transaction log

A system table (*syslogs*) in which all changes to the database are recorded.

trigger

A special form of **stored procedure** that goes into effect when a user gives a change command such as **insert**, **delete**, or **update** to a specified table or column. Triggers are often used to enforce referential integrity.

user authorization identifier

User authorization identifiers are associated with each schema. All the objects are said to be owned by or to have been created by the associated user authorization identifier for the schema.

user id

The ID number by which a user is known in a specific database. Distinct from server user ID.

view

An alternative way of looking at the data in one or more tables. Usually created as a subset of columns from one or more tables.

write access

Permission to write an object (for example, to update a row or to add a row to a table).

Index

The index is divided into two sections:

- Symbols

  Indexes each of the symbols used in Sybase SQL Server documentation.

- Subjects

  Indexes subjects alphabetically.

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

Symbols

A

B

C

D

O

P

R

S

T

U

V

W