Introducing Sybase® SQL Server™ for Workplace UNIX

Sybase SQL Server Release 11.0.x

Document ID: 34968-01-1101-01

Last Revised: April 15, 1996

Principal author: Sara Van Dyke

Document ID: 34968-01-1101

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.

Copyright © 1989–1996 by Sybase, Inc. All rights reserved.

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

Sybase Trademarks

Sybase, the Sybase logo, APT-FORMS, Data Workbench, DBA Companion, Deft, GainExposure, Gain *Momentum*, Navigation Server, PowerBuilder, Powersoft, Replication Server, S-Designor, SQL Advantage, SQL Debug, SQL SMART, SQL Solutions, SQR, Transact-SQL, and VQL are registered trademarks of Sybase, Inc. ADA Workbench, AnswerBase, Application Manager, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, APT Workbench, Backup Server, Bit-Wise, Client-Library, Configurator, Connection Manager, Database Analyzer, DBA Companion Application Manager, DBA Companion Resource Manager, DB-Library, Deft Analyst, Deft Designer, Deft Educational, Deft Professional, Deft Trial, Developers Workbench, DirectCONNECT, Easy SQR, Embedded SQL, EMS, Enterprise Builder, Enterprise Client / Server, Enterprise CONNECT, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, ExElerator, Gain Interplay, Gateway Manager, InfoMaker, Interactive Quality Accelerator, Intermedia Server, Maintenance Express, MAP, MDI, MDI Access Server, MDI Database Gateway, MethodSet, Movedb, Navigation Server Manager, Net-Gateway, Net-Library, New Media Studio, ObjectCONNECT, OmniCONNECT, OmniSQL Access Module, OmniSQL Gateway, OmniSQL Server, OmniSQL Toolkit, Open Client, Open Client CONNECT, Open Client / Server, Open Client / Server Interfaces, Open Gateway, Open Server, Open Server CONNECT, Open Solutions, PC APT-Execute, PC DB-Net, PC Net Library, Powersoft Portfolio, Powersoft Professional, Replication Agent, Replication Driver, Replication Server

Manager, Report-Execute, Report Workbench, Resource Manager, RW-DisplayLib, RW-Library, SAFE, SDF, Secure SQL Server, Secure SQL Toolset, SKILS, SQL Anywhere, SQL Code Checker, SQL Edit, SQL Edit / TPU, SQL Server, SQL Server / CFT, SQL Server / DBM, SQL Server Manager, SQL Server Monitor, SQL Station, SQL Toolset, SQR Developers Kit, SQR Execute, SQR Toolkit, SQR Workbench, Sybase Client / Server Interfaces, Sybase Gateways, Sybase Intermedia, Sybase Interplay, Sybase IQ, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase Virtual Server Architecture, Sybase User Workbench, SyBooks, System 10, System 11, the System XI logo, Tabular Data Stream, Warehouse WORKS, Watcom SQL, web.sql, WebSights, WorkGroup SQL Server, XA-Library, and XA-Server are trademarks of Sybase, Inc.

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

Restricted Rights

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

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

Table of Contents

About This Book

Introducing Sybase SQL Server for Workplace UNIX introduces the features and capabilities of Sybase® SQL Server™ for Workplace UNIX, an integrated set of Sybase products for developing client/server database applications.

Audience

This manual provides a starting point for:

- System administrators and other privileged users, who perform administrative tasks for the SQL Server for Workplace UNIX installation as a whole
- Database owners, who create and manage databases
- Application developers, who write programs to access the information in SQL Server databases

How to Use This Book

Introducing Sybase SQL Server for Workplace UNIX includes information you need to begin using SQL Server for Workplace UNIX. For each topic, the manual tells you where to get more detailed information if you need it.

Introducing Sybase SQL Server for Workplace UNIX is organized according to the tasks that users perform. The manual contains the following chapters:

Chapter	Topics Covered	Primary Audience
Chapter 1, "What Is Sybase SQL Server for Workplace UNIX?"	Introductory topics and concepts about SQL Server for Workplace UNIX	System administrators, database owners, application developers
Chapter 2, "Managing a SQL Server Installation"	Controlling system resources, managing users, tuning for performance, and performing other system maintenance activities	System administrators

Chapter	Topics Covered	Primary Audience
Chapter 3, "Managing SQL Server Data"	Creating and managing databases and database objects	Database owners
Chapter 4, "Backing Up and Restoring Data"	Protecting your data from a system or media failure	System administrators
Chapter 5, "Accessing Data with Transact-SQL"	Using Sybase's enhanced Structured Query Language to access and manipulate the information in your SQL Server databases	System administrators, database owners, application developers
Chapter 6, "Building Client / Server Applications"	Using the Open Client™ application programming interfaces to build applications that access SQL Server databases	Application developers
Glossary	Defines terms relevant to SQL Server for Workplace UNIX	System administrators, database owners, application developers

Related Documents

SQL Server for Workplace UNIX includes a full range of documentation, in printed and online form, that provides comprehensive information to help you make the best use of the product's features and capabilities.

At the end of each chapter in this manual is a list of documents that provide additional information about the topics discussed within the chapter.

Conventions Used in This Manual

The following conventions in typography and terminology are used in this manual for consistency and clarity.

SQL Syntax Conventions

Listed below are the syntax conventions that this manual uses.

Key	Definition
command	Command names, command option names, utility names, utility flags, and other keywords are in **`bold Courier`** in syntax statements, and in **bold Helvetica** in paragraph text.
variable	Variables, or words that represent values that you supply, are in italics.
{ }	Curly braces indicate that you choose at least one of the enclosed options. (Do not type the braces.)
[ ]	Brackets indicate that you can include none or any of the enclosed options. (Do not type the brackets.)
( )	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 to be typed as part of the command

The manual also observes the following conventions:

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

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

  In syntax statements, keywords (commands) are in normal font and identifiers are in lowercase; normal font is used for keywords, italics for user-supplied words.

- Examples showing the use of Transact-SQL® commands are printed as follows:

  ```
  select * from publishers
  ```

- You can disregard case when you type keywords:

  "SELECT" is the same as "Select" or "select."

- Client-Library™ routine syntax is shown as follows:

```
CS_RETCODE ct_init(context, version)

CS_CONTEXT context;
CS_INT version;
```

- Client-Library structure names and symbolic constants are shown in uppercase:

  CS_CONTEXT, CS_SYNC_IO

Formatting SQL Statements

Transact-SQL has no rules about line length or breaks. However, for readability, examples and syntax statements in this manual are formatted so that each clause begins on a new line. Clauses that have more than one part extend to additional lines, which are indented.

1 What Is Sybase SQL Server for Workplace UNIX?

Introduction

This chapter provides background information to help you use Sybase SQL Server for Workplace UNIX effectively. The chapter includes:

- A description of Sybase SQL Server for Workplace UNIX and its components
- An overview of how to use SQL Server for Workplace UNIX
- Background information about:
  - Structured Query Language (SQL)
  - Relational databases in general
  - System databases and tables that are provided with SQL Server
- A discussion of the roles, activities, and permissions involved in using SQL Server
- Directions for getting more information about using SQL Server for Workplace UNIX

Sybase SQL Server for Workplace UNIX is an integrated set of Sybase relational database products for developing and deploying client/server applications on desktop platforms. SQL Server for Workplace UNIX brings the power of Sybase enterprise client/server computing to departmental, workgroup, and small business users.

This table shows the products that SQL Server comprises:

Product	Description
SQL Server	The Sybase high-performance relational database that features an advanced multithreaded architecture, server-enforced integrity, and high transaction and query throughput for multiple users.
Backup Server™ (installed as part of SQL Server)	A control server that operates concurrently with SQL Server and features high-speed, online backup, loading, and recovery capabilities.

Product	Description
SQL Server Monitor™	SQL Server monitoring tool consisting of four components. SQL Server Monitor Server captures performance data about SQL Server and databases. SQL Server Monitor Historical Server stores performance data for deferred inspection and analysis. SQL Server Monitor Client is a Windows application that displays performance data captured by Monitor Server. SQL Server Monitor Client Library is an application programming interface that enables you to build your own SQL Server monitoring applications.
SQL Server Manager™ for Windows	A Windows tool to administer SQL Server through a graphical user interface.
Open Client™	The Client-Library (and DB-Library™) programming interface that provides applications, third-party products, and other Sybase products with library functions to communicate with SQL Server. Includes Net-Library™, which provides network protocol services to support for the Open Client/SQL Server connections on many desktop platforms.
ODBC Driver Kits	Application programming interfaces (API) that let you access SQL Server data from 16-bit or 32-bit Windows applications and development tools.
SyBooks™	A collection of SQL Server for Workplace UNIX documentation and a browser for searching and viewing the documentation online.

Sybase SQL Server for Workplace UNIX: An Overview

The table below describes the high-level tasks involved in using SQL Server for Workplace UNIX to organize and access information.

Activities are performed by various categories of user, as shown in the table. You will learn more about these user categories, or roles, in the section "Roles in SQL Server," later in this chapter.

Activity	When Performed	Responsible Role	For More Information
Installing or upgrading SQL Server for Workplace UNIX products	Before beginning to use SQL Server for Workplace UNIX	System Administrator	*Installing Sybase SQL Server for Workplace UNIX*
Managing SQL Server • Configuring SQL Server • Adding and enabling users • Managing physical resources • Monitoring SQL Server database activity and tuning SQL Server for best performance	After installing SQL Server; also, as needed due to changing conditions and needs at your site	System Administrator, System Security Officer	Chapter 2, "Managing a SQL Server Installation" and: • *Configuring Sybase SQL Server* for your platform • *SQL Server System Administration Guide* • *SQL Server Monitor Client User's Guide for Microsoft Windows* • *SQL Server Monitor User's Guide* • *SQL Server Monitor Historical Server User's Guide* • *SQL Server Performance and Tuning Guide*
Creating SQL Server databases, tables, and other database objects	When setting up SQL Server to store user data; also, as needed to redefine how the data is stored or to add, update, or delete databases, tables, and database objects	Database Owner	Chapter 3, "Managing SQL Server Data," and: • *Transact-SQL User's Guide* • *SQL Server Manager User's Guide*

Activity	When Performed	Responsible Role	For More Information
Backing up and recovering the data in the SQL Server databases	For backing up, periodically, while using SQL Server; for recovering, after a system crash, media failure, or other loss of data	System Administrator, Operator	Chapter 4, "Backing Up and Restoring Data," and: • *SQL Server System Administration Guide* • *SQL Server Manager User's Guide*
Issuing Transact-SQL queries to SQL Server	When accessing, updating, or deleting SQL Server data; also, when performing database and system maintenance activities	SQL Server end users, Database Owner, System Administrator	Chapter 5, "Accessing Data with Transact-SQL," and: • *Transact-SQL User's Guide* • *SQL Server Reference Manual*
Building client applications to access data on SQL Server	When creating client applications for end users to access SQL Server data on an ongoing basis	Open Client Application Developers	Chapter 6, "Building Client/Server Applications," and: • *Open Client Client-Library/C Programmer's Guide* • *Open Client Client-Library/C Reference Manual* • *Sybase System 10 ODBC Reference Guide*

What Is a Relational Database Management System?

In a relational database management system (RDBMS), data is represented as tables, also known as relations. A database is made up of a set of related tables.

Each row of a table describes one occurrence of an entity, such as a person, a product, or a sale. Each column describes one characteristic of the entity, such as a person's name or address, a product's price, or the date of a sale.

Figure 1-1 shows how an RDBMS stores data in a table.

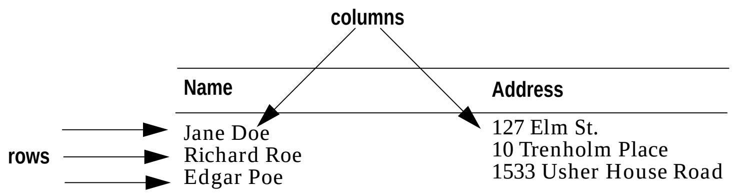

Figure 1-1: A table in a relational database

Databases and Database Objects

An RDBMS database comprises tables of related information. For example, the *pubs2* sample database that comes with SQL Server for Workplace UNIX represents information about a book distributing business. One table in this database contains information about authors, another table contains records of book sales, and other tables represent other aspects of the book distribution business.

In addition to tables, SQL Server databases contain other objects that you use to store or manipulate data. The table below describes the objects that the SQL Server databases can contain.

Object	Description	For Information
Table	A collection of rows and columns containing related data items. There are two types of table: user tables and system tables.	"Creating, Altering, and Dropping Tables" on page 3-2
Rule	Specifies what values users are allowed to enter in a specific column.	"Rules" on page 3-8
Default	A value that SQL Server inserts into a column if the user does not explicitly enter a value.	"Default Values" on page 3-8

Object	Description	For Information
Stored Procedure	A collection of SQL statements that you can issue as a single command. Using stored procedures improves SQL Server performance because they are stored in a preprocessed form. SQL Server provides a set of stored procedures, called system procedures to be used for system administration and other SQL Server maintenance tasks.	"Stored Procedures" on page 5-13
Trigger	A type of stored procedure that goes into effect when you insert, delete, or update data in a specified table.	"Triggers" on page 3-9
View	A virtual table derived from columns and rows of one or more tables (or other views). The plan for creating a view is stored by SQL Server, not the view itself.	"Views" on page 3-6
Constraint	There are two types of constraint: • Referential integrity: requires that data changed in one table is also changed in another table. (For example, when a new book title is entered in the *salesdetail* table it is also inserted in the *titles* table.) • Check integrity: validates that data inserted into a column of a table is a legal value. (For example, a check integrity constraint might ensure that all customer discounts are within a specified limit.)	"Data Integrity" on page 3-7

What Is Transact-SQL?

SQL (Structured Query Language) is a high-level language for accessing data stored in relational database systems. SQL has been approved as the official relational database query language standard by the American National Standards Institute (ANSI) and the International Standards Organization (ISO).

Transact-SQL is the Sybase version of Structured Query Language. Transact-SQL is compatible with IBM SQL and most other commercial implementations of SQL and provides important extra capabilities and functions not defined by the ANSI standard. SQL includes statements for querying databases, as well as creating databases and database objects, inserting data, modifying existing data, and other functions.

Getting Statement Syntax Information

You can install the *sybsyntax* system database, which contains syntax information for Transact-SQL statements, system procedures, utilities, and other routines. To access the information in the *sybsyntax* database, use the **sp_syntax** system procedure. You can specify all or part of a statement name, and SQL Server returns the syntax for using that statement.

System Databases, Tables, and Procedures

Just as you organize your own information into databases and tables, SQL Server organizes the information that it uses to manage user data into system databases and system tables.

System Databases

SQL Server includes the system databases described in this table:

Name	Description
master	Controls user databases as well as the operation of SQL Server as a whole.
sybsystemprocs	Stores SQL Server system and user-created stored procedures that can be used to query the tables in the system databases.
model	Provides a database template that is copied each time a user creates a user database. Any changes that you make to this database are reflected in each new database created by a user.
tempdb	Provides storage that SQL Server uses for temporary tables and other temporary working storage needs (for example, intermediate results of complex queries).
sybsecurity	Contains the audit system for SQL Server (optional).
sybsyntax	Contains syntax descriptions for Transact-SQL statements and SQL Server commands (optional).
pubs2	A sample database (optional).

System Tables

The *master* database, created when you install SQL Server, is made up entirely of system tables containing information to keep track of SQL Server as a whole.

Each user database contains a subset of the system tables to keep track of information specific to that database. The system tables are also called the data dictionary or the system catalogs.

Querying the System Tables

Use Transact-SQL to query the system tables just as you would any other SQL Server table. In this way, you can get information about the contents and operations of SQL Server and your databases.

For example, the following SQL query returns the names of all databases on SQL Server:

```
select * from sysdatabases
```

Data in the system tables is inserted, updated, or deleted by Transact-SQL statements or by system procedures. You should normally not modify the system tables.

System Procedures

SQL Server supplies system procedures for you to use as:

- Shortcuts for retrieving information from the system tables
- Mechanisms for accomplishing database administration and other activities that involve updating system tables

The system procedures that SQL Server supplies constitute one of its most powerful features. You can use a single procedure to accomplish complex sets of activities related to updating the system tables and getting data from them. There are system procedures for such activities as adding users, auditing system activity, and configuring SQL Server.

SQL Server creates the system procedures in the *sybsystemprocs* database during the installation process. The System Administrator owns this database. The names of all system procedures begin with "sp_".

One example of a system procedure is **sp_dboption**, which returns the current values of the SQL Server options. These options can be configured after SQL Server is installed on your server.

The *pubs2* Sample Database

The *pubs2* sample database is the basis of most of the examples in the SQL Server for Workplace UNIX documentation. Use the *pubs2* database to practice the concepts you learn while using the products and the documentation.

The *pubs2* database represents information for a book distributing business. It contains a guest user mechanism that allows any authorized SQL Server user to access it and to view, update, insert, and delete data. Guest accounts are discussed in “The Guest Database User” on page 2-5.

Roles in SQL Server

SQL Server uses the concept of roles for assigning permissions to users to perform certain administrative tasks and functions. Roles are granted to individual server login accounts, and actions performed by these users can be audited and attributed to them.

There are three special SQL Server roles that have certain administrative tasks associated with them:

- System Administrator
- System Security Officer
- Operator

Users who will perform administrative tasks on SQL Server should have appropriate roles assigned to their individual login accounts. More than one SQL Server login account can be granted a particular role, and one account can possess more than one role.

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. 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
- Transferring bulk data between SQL Server and other software programs
- 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 user databases and granting ownership of them
- Granting and revoking the System Administrator role
- Setting up groups (which are convenient for granting and revoking permissions)

A System Administrator takes on the identity of Database Owner when accessing any database, including *master*. There are several activities that only a System Administrator can perform.

System Security Officer

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

- 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 activities that only a System Security Officer can perform and for which permissions cannot be transferred to other users.

Operator

An operator is a user who can back up and load databases on a server-wide basis. The Operator role allows a single user to back up and restore all databases on a SQL Server without having to be the owner. These operations can be performed in a single database by the Database Owner and the System Administrator.

The “sa” Login Account

When first installed, SQL Server comes with a single login account, known as “sa”. Log into the “sa” account to begin performing administrative tasks, such as setting up users and assigning roles.

A user who logs into the “sa” account has wide privileges throughout SQL Server, since it is configured for both the System Administrator and System Security Officer roles. This account should not be used regularly by any user.

Data Ownership Roles

In addition, SQL Server recognizes two kinds of object owners, who gain special status because of the objects they own. These ownership types are:

- The owner of a user database
- The owner of a database object

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. The owner of a database may:

- Allow other SQL Server users access to the database
- Give other users permission to create objects and execute statements within the database

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.

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.

Getting More Information

Sources of information for topics discussed in this chapter are listed below. For information on where these topics are discussed in this book, refer to the index.

Topic	Source(s) of Information
Client-Library	• *Open Client Client-Library/C Reference Manual* • *Open Client Client-Library/C Programmer's Guide*
Client / server application development	• *Open Client Client-Library/C Reference Manual* • *Open Client Client-Library/C Programmer's Guide*
Database objects	• *Transact-SQL User's Guide*
DB-Library	• *Open Client DB-Library/C Reference Manual*
pubs2 sample database	• *SQL Server Reference Supplement* • *SQL Server System Administration Guide*
Roles	• *SQL Server Security Features User's Guide* • *SQL Server Security Administration Guide*
Syntax for Transact-SQL	• *Transact-SQL User's Guide* • *SQL Server Reference Manual*
System databases and system tables	• *SQL Server System Administration Guide* • *SQL Server Reference Supplement*
System procedures	• *SQL Server Reference Manual*
User permissions	• *SQL Server Security Administration Guide* • *SQL Server Security Features User's Guide*

2 Managing a SQL Server Installation

Introduction

This chapter discusses system-wide activities that you as System Administrator, or other designated users, perform while managing the SQL Server installation, including:

- Configuring SQL Server
- Managing SQL Server users
- Managing SQL Server's physical resources
- Customizing and fine tuning SQL Server
- Auditing SQL Server user activity

SQL Server Manager

You can perform many of the activities described in this chapter interactively with SQL Server Manager, a Windows application for managing SQL Server. Use SQL Server Manager to:

- Manage disk storage
- Set up SQL Server user accounts
- Create databases and database objects
- Monitor system performance
- Troubleshoot server problems
- Start and stop SQL Server
- Fine-tune SQL Server by changing server parameters

Using SQL Server Manager, you can accomplish SQL Server management, administrative, and maintenance functions quickly and easily.

Configuring SQL Server

sybinit

After you install SQL Server, you configure it using the configuration utility, **sybinit**. See *Configuring Sybase SQL Server* for your platform for instructions on using **sybinit**.

SQL Server Backup Server

Backup Server is provided as part of SQL Server for Workplace UNIX. It is used for maintaining backup copies of your databases and transaction logging files. To use Backup Server, you need to activate it explicitly.

Localizing SQL Server

SQL Server supports international use through files that contain translated error messages, character set definitions, sort order definitions, and utilities for converting SQL Server's character set into the appropriate character set for a PC or terminal. These files are called localization files, and they are included when you install any Language Module with your SQL Server.

SQL Server Logins and Users

For users to access data stored on SQL Server, they must have:

- A SQL Server login account with its associated login ID and password
- Explicitly-granted access to one or more databases
- Explicitly-granted permissions for performing various actions on the database and database objects

Users may be organized into groups and permissions granted to the group. This mechanism allows similar types of users to be granted data access capabilities at one time, rather than one by one.

Activities related to setting up and managing SQL Server users are carried out by the System Security Officer, the System Administrator, and the Database Owner.

The table below summarizes the activities and who performs them.

Activity	Executed By
Create new logins, assign default database and default language	System Security Officer
Create groups	Database Owner or System Administrator
Add users to database, assign aliases, assign groups	Database Owner or System Administrator
Grant permissions on statements and database objects to groups or users	Database Owner, System Administrator, or database object owner

Database Users

Once an individual has been given a SQL Server login, that person may then become a database user for any user database on that server. Although a SQL Server login does not automatically grant the user access to a database, a default database may be assigned to a SQL Server login account that allows the user to directly access that database upon logging in.

After a user has been given access to a database, the owner of the database or the owner of objects within the database still must give the user permission to read, modify or delete data, use certain Transact-SQL statements, and execute stored procedures.

The System Security Officer sets up users and groups for individual databases on SQL Server. The System Security Officer also grants permissions to users and assigns a default database and default language for users. Users and their permissions are assigned for individual databases, one at a time.

Use the SQL Server Manager to manage SQL Server logins and database users and groups.

For a small installation, you can assign both System Security Officer and System Administrator roles to the same user in order to consolidate database user management activities with a single person. For large installations with many users, it is recommended that System Security Officer and System Administrator functions be separate and carried out by different users.

Passwords

Passwords help prevent unauthorized access to SQL Server data. SQL Server passwords must be at least six characters long and can contain any printable letters, numerals, or symbols.

When creating 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 and punctuation, for example, "4A$sk."

Users can change their passwords at any time.

Groups

A group is a mechanism for assigning a collective name to multiple database users for the purpose of granting and revoking permissions. Using groups, you can manage permissions for many users in a single operation.

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

A System Administrator or Database Owner can create a group at any time. The System Administrator assigns or reassigns users to groups.

Visitor Login Accounts

To accommodate occasional users of SQL Server, the System Security Officer can create a login name (for example, "visitor") and password that visiting users can use. Typically, such users are granted very restricted permissions.

Note that visitor accounts allow more than one person to log in to SQL Server on a single account. This results in poor auditing records, since you cannot distinguish the actions of individual users of the visitor account. To avoid this situation, assign each user a separate login account.

Remote Users

You can enable remote access to allow users on another SQL Server to execute stored procedures on your server. You can also set up users of your SQL Server to execute remote procedure calls to the remote server.

To enable remote procedure calls, both the local and the remote server must be specifically configured. The System Security Officer and the System Administrator share control over remote user access and any associated security issues. Database Owners are responsible for granting permissions to remote users.

The Guest Database User

A Database Owner can create a database user named “guest”. Adding a guest user to a database allows an owner to permit all SQL Server users to use that database without having to name each one as a database user explicitly.

When the user name “guest” is first added to a database, “guest” inherits the privileges of the default group “public”. The Database Owner and the owners of database objects can change these permissions to make the privileges of “guest” more or less restrictive.

Locking SQL Server Logins

For security reasons, you can lock a SQL Server account to prevent a particular user from logging in. To ensure that someone always has access, SQL Server maintains at least one unlocked System Security Officer account and one unlocked System Administrator account.

Getting Information About Users

SQL Server Manager displays information about users, such as:

- Logins mapped to a user
- Objects on which users have permissions
- Objects owned by a user
- Users and aliases mapped to a login
- Users in a group

Additionally, Sybase-provided stored procedures report the following information about users:

- Currently active SQL Server users and processes
- Login accounts
- Users and aliases in the database
- Groups within a database

Changing User Information

The table below summarizes ways in which user information can be changed and who can make the changes.

Activity	Executed By
Change another user’s password	System Security Officer
Change own password	User
Change group assignment of a user	System Administrator, Database Owner
Change a login account’s default database, default language, or full name	System Administrator
Change own default database, default language, or full name	User

Aliases

You can use an alias to treat more than one person as the same user, with the same privileges, inside a database. In particular, several users can assume the role of Database Owner.

For example, you might create an alias called “vp” to be used by all vice presidents, each of whom has an individual SQL Server account linked to that alias.

You also can use an alias to set up a collective user identity, within which the identities of individual users can be traced by auditing their activities. You cannot do this with a visitor or guest account.

Managing Physical Resources

As System Administrator, you are responsible for managing the physical resources used by SQL Server.

Database Devices

You must prepare database devices before you can create user databases to store information. This process is called initialization.

A database device stores the objects that make up databases. The term “device” does not necessarily refer to a distinct physical device. A database device is any piece of a disk (such as a partition or a file in the file system) used to store databases and database objects.

Databases are allocated storage space when you create or enlarge them. When creating a database, you can specify one or more database devices and the amount of space on each device to be allocated to the new database.

Management Tasks

A System Administrator initializes new database devices. After initializing the devices, the System Administrator creates a pool of default database devices to be used by all SQL Server users when creating databases. Whenever users create (or enlarge) databases without specifying a database device, new disk space is allocated from this pool.

SQL Server uses default values for many aspects of its storage management—where databases, tables, and indexes are placed and how much space is allocated for each of them. However, the System Administrator can modify any of these values.

After stabilizing an application and determining its data-handling requirements, you refine storage management to improve performance. These refinements are not an essential aspect of data definition, but rather a means of improving performance.

Some of the activities associated with management of physical resources are:

- Making a physical device available to a specific server
- Assigning a database device name
- Making a database device available as the default device

- Mirroring (duplicating) a database device
- Placing the database's logs on the default device or on a specific device
- Allocating space on database devices
- Creating a segment (a named subset of database devices) from the devices available to a specific database
- Creating a table or index on a specific segment or on the default device

Database Size

If you do not specify a size when creating a database, the database is created with the default amount of space. This amount will be the greater of these two values:

- The default **database size** configuration parameter
- The size of the *model* database

You can allocate more space for the *model* database as well as reset the **database size** configuration parameter.

It is difficult to reclaim storage space after it has been assigned. You cannot deallocate space that has been assigned to a database unless you drop the database first; however, you can always add more space.

When a database grows to fill all of the allocated space, you can add space for database objects or the transaction log, or both. Permission to enlarge a database defaults to the Database Owner.

Location and Size of the Transaction Log

The transaction log is a system table that records all changes made to a database. By default, the transaction log is located on the same device as the database, but unless you are creating very small, non-critical databases, you should always place the transaction log on a separate device.

Designating a separate device for the transaction log has the following benefits:

- SQL Server performance is improved because writes to the database and to the transaction log occur concurrently rather than serially.

- Full recovery is guaranteed if the hard disk crashes.
- You only need to back up the transaction log as a recovery mechanism rather than always backing up the entire database, thus saving time and tapes.
- You can establish a fixed size for the log, keeping it from competing with regular database activity for space.
- SQL Server creates default free-space threshold monitoring on the log segment, allowing you to create additional free-space monitoring on the log and data portions of the database.

The size of the device required for the transaction log varies according to the amount of update activity and the frequency of transaction log dumps. In general, allocate to the log 10 to 25 percent of the space you allocate to the database itself.

Database Segments

Segments are named subsets of the database devices available to a specific database. A named segment functions as a label that points to one or more database devices. Use segment names when creating databases or indexes to place tables or indexes on specific database devices.

Using segments can improve SQL Server performance and give increased flexibility over placement, size, and space usage of specific database objects. For example:

- If, by using segments, a table is placed on one device and its index is on a device on another disk controller, disk head travel, and thus the time required to read or write to the disk, can be reduced.
- If you use segments to split a large, heavily used table across devices on two separate disk controllers, read and write times can improve.
- If you place tables and indexes on specific segments, the space available to the devices is defined by those segments, and other objects cannot contend with them for that space. If necessary, segments can be extended to include additional devices.
- When you specify database segments, you can establish thresholds that trigger a warning when space becomes low on a specific database segment.

You can use SQL Server Manager to create and manage segments. Or you can use Sybase-provided system stored procedures, if you prefer.

Disk Mirroring

Mirroring a disk duplicates the database; all writes to the device are copied to a separate physical device. If one of the devices fails, the other contains an up-to-date copy of all transactions. Disk mirroring can provide recovery of SQL Server databases without downtime in the event of media failure.

When a read or write to a mirrored device fails, the bad device becomes unmirrored, and SQL Server displays an error message. SQL Server continues to run, unmirrored.

Deciding Whether to Implement Mirroring

The decision about whether to implement disk mirroring should be based on your analysis of the trade-offs described in this section. You can also consider mirroring just the transaction logs, all devices on a server, or selected devices.

Disk mirroring involves the following cost and performance trade-offs:

- Impact on performance

  Mirroring user databases increases the time needed to write transactions because they are written to two disks.

- Cost of downtime

  If disk mirroring is not implemented, recovery of SQL Server in the event of media or power failures causes system downtime that will impact productivity of your users.

- Speed of recovery

  You can achieve recovery without downtime when the *master* and user databases (including logs) are mirrored, and you can recover without the need to reload transaction logs.

- Storage space

  Immediate recovery requires full redundancy (all databases and logs mirrored); this consumes disk space.

Using SQL Server Manager, you can issue any of the disk mirroring commands while the devices are in use, including starting or stopping the mirroring operation.

Customizing and Fine-Tuning SQL Server

The System Administrator, System Security Officer, or Database Owner can monitor the performance of SQL Server and customize and fine-tune its operation.

The variables that affect SQL Server and database behavior are:

- Configuration parameters, which affect the environment for the entire SQL Server
- Database options, which affect the behavior of a single database
- Transact-SQL **set** statements, which affect the current user session or an executing stored procedure or trigger

SQL Server for Workplace UNIX provides the following applications for performance tuning:

- SQL Server Manager to change SQL Server configuration parameters and database settings
- SQL Server Monitor to monitor and report information on SQL Server performance

SQL Server for Workplace UNIX also provides a query tool, **wisql**, which you can use to execute stored procedures that change configuration parameters and database options.

Configuration Parameters

Configuration parameters control various aspects of SQL Server's operating environment. When you install SQL Server, default values for these parameters are established, based on assumptions about how SQL Server is generally used. You can change the values of these parameters to fine-tune the performance of SQL Server.

Configuration parameters can be grouped according to the area of SQL Server behavior they affect:

- Backup and recovery
- Cache management
- Disk I/O

- Languages
- Lock management
- Memory use
- Network communications
- Operating system resources
- Physical memory
- Processors
- SQL Server administration
- User environment
- General characteristics

System Administrators and System Security Officers can modify configuration parameters using SQL Server Manager or **wisql**.

Database Options

Database options control many aspects of database behavior, such as:

- The behavior of transactions
- Defaults for table columns
- Restrictions to user access
- Performance of recovery and bulk copy operations
- Log behavior

System Administrators and Database Owners can configure the database option settings for a SQL Server database using SQL Server Manager or **wisql**.

The *set* Statement

The Transact-SQL **set** statement tells SQL Server how to handle queries and stored procedures. These query processing options are set for the duration of the user's work session. If a stored procedure contains a set statement, its effect lasts while the stored procedure executes. No permissions are required to use the **set** statement.

For example, the following statement causes SQL Server to stop processing queries after it returns the first ten rows:

```
set rowcount 10
```

You can execute the **set** statement from **wisql**.

Monitoring SQL Server Activity

SQL Server for Workplace UNIX provides two facilities for monitoring SQL Server activity: SQL Server Monitor and a set of predefined global variables.

SQL Server Monitor

SQL Server Monitor is a tool for gathering and displaying performance data. SQL Server Monitor helps you evaluate SQL Server operation, isolate the causes of performance problems, and gather statistics for capacity planning.

Monitor Components

SQL Server Monitor consists of four components:

- Monitor Server is an Open Server™ application that reads SQL Server performance data from shared memory. Monitor Server collects data in a manner that has a minimal impact on SQL Server performance.
- Monitor Client is a Windows application that displays performance data generated by Monitor Server.
- Monitor Historical Server is an Open Server application that records SQL Server performance data for deferred analysis.
- Monitor Client Library is an application programming interface that provides routines for acquiring and integrating performance data from Monitor Server and SQL Server. You can use this API for building custom SQL Server monitoring applications.

What Is Monitored?

SQL Server Monitor provides system information about

- Cache management
- Device I/O
- Memory allocation
- Network activity
- Object lock status

- Object page I/O
- Performance trends
- Process activity
- Process detail
- Process lock activity
- Stored procedure activity
- Transaction activity

Advantages of SQL Server Monitor

SQL Server Monitor helps you manage SQL Server by:

- Presenting an overall performance picture
- Establishing a performance baseline
- Pinpointing the nature of problems
- Giving feedback on the effect of tuning activities
- Providing session and display management features

Monitoring with Global Variables

SQL Server uses a series of pre-defined global variables that hold system-supplied values that track SQL Server activity. These variables are distinguished from local variables by having two @ signs preceding their names (for example, *@@error*). The global variable *@@error*, as an example, contains the number of the last error message generated by SQL Server.

You can find the values of these variables in either of two ways:

- Use the **sp_monitor** system procedure
- Query the variables directly

Execute the **sp_monitor** procedure to return the current values of global variables. You can monitor how they have changed since the last time you ran the procedure.

Auditing SQL Server Activity

SQL Server can generate an audit trail to record system activity. The audit trail can detect all activity within SQL Server and its databases.

By examining the audit trail, the System Security Officer can inspect patterns of access to objects in databases and can monitor the activity of specific users. In the role of System Security Officer, you can:

- Execute the auditing system procedures
- Read the audit trail
- Access the audit database

The SQL Server audit system consists of these components:

- The auditing *(sybsecur.dat)* database, which is created using the Server Config utility. This database contains two important system tables used specifically for auditing:
  - *sysaudits*, which contains the audit trail
  - *sysauditoptions*, which contains the global auditing choices
- The audit queue, which holds information about an audited event in memory. The audit process then adds the information to the audit trail.

Auditing Options

You can audit the following SQL Server activities:

- Successful or failed login attempts by all users or by individual users, including remote logins
- Logouts from SQL Server, including unintentional logouts such as dropped connections
- Restarting of SQL Server
- Turning roles on and off
- The use of statements requiring a specific role for execution
- Fatal errors (errors that disconnect from SQL Server and require the client program to be restarted), nonfatal errors, or both kinds
- Attempts to access tables and views
- Use of certain statements within a database
- Execution of stored procedures and triggers

Getting More Information

Sources of information for topics discussed in this chapter are listed below.

Topic	Source(s) of Information
Aliases	• *SQL Server System Administration Guide*
Auditing SQL Server	• *Configuring Sybase SQL Server for your platform* • *SQL Server System Administration Guide*
Configuration, default installation	• *Installing Sybase SQL Server for Workplace UNIX* • *Configuring Sybase SQL Server for your platform* • *SQL Server System Administration Guide*
Configuration parameters	• *SQL Server System Administration Guide* • *SQL Server Manager User's Guide*
Configuring SQL Server	• *Configuring Sybase SQL Server for your platform* • *SQL Server System Administration Guide*
Database devices	• *Configuring Sybase SQL Server for your platform* • *SQL Server System Administration Guide*
Default devices	• *Configuring Sybase SQL Server for your platform* • *SQL Server System Administration Guide*
Enlarging database	• *SQL Server System Administration Guide*
Error log	• *Configuring Sybase SQL Server for your platform*
Groups	• *SQL Server Security Administration Guide*
Indexes	• *SQL Server System Administration Guide*
Initializing SQL Server	• *SQL Server System Administration Guide*
Installing and configuring SQL Server	• *Installing Sybase SQL Server for Workplace UNIX* • *Configuring Sybase SQL Server for your platform* • *SQL Server System Administration Guide*

Topic	Source(s) of Information
Localization	• *SQL Server System Administration Guide* • *Transact-SQL User's Guide*
Location of database	• *Configuring Sybase SQL Server for your platform* • *SQL Server System Administration Guide*
Locking a user login	• *SQL Server Security Administration Guide*
Monitoring SQL Server	• *SQL Server Monitor Client User's Guide for Microsoft Windows* • *SQL Server Monitor User's Guide* • *SQL Server Monitor Historical Server User's Guide* • *SQL Server Monitor Client Library Programmer's Guide* • *Configuring Sybase SQL Server for your platform* • *SQL Server System Administration Guide*
Performance optimization	• *SQL Server Performance and Tuning Guide*
Remote stored procedure calls	• *SQL Server Security Administration Guide* • *SQL Server Reference Manual*
Resource management	• *SQL Server System Administration Guide* • *SQL Server Performance and Tuning Guide*
SQL Server Manager	• *SQL Server Manager User's Guide*
Transaction log	• *SQL Server System Administration Guide*
User information	• *SQL Server Manager User's Guide* • *SQL Server Reference Manual* • *Transact-SQL User's Guide*
Users	• *SQL Server System Administration Guide*
Utilities	• *SQL Server Utility Programs for UNIX*

3 Managing SQL Server Data

Introduction

This chapter provides information about managing SQL Server data. It describes:

- Creating and removing databases
- Creating, altering, and dropping tables
- Creating views from columns or rows of specified tables
- Creating indexes for quicker access to data
- Controlling user access to data
- Copying databases and tables

Creating and Removing Databases

A user having the System Administrator role may grant permission to other users to create databases. In many SQL Server installations, only System Administrators can create databases, so they can control database placement and database device allocation. In these situations, System Administrators create new databases on behalf of other users and then transfer ownership to those users.

All users should consider developing scripts to automatically re-create databases, tables and other database objects, especially stored procedures. In the event of database corruption resulting from media or power failure, the scripts can rebuild these objects quickly and accurately.

User Databases

SQL Server uses the *model* database as a template for new user-created databases in SQL Server. When a new database is created, SQL Server makes a copy of the *model* database.

The *model* database contains the required system tables for a user database. A new database is created the same size as or larger than *model*, but it cannot be smaller. As System Administrator, you can update *model*, as you would any other database, to add:

- Tables and other database objects

- Stored procedures
- Datatypes
- System parameters
- User names
- Specific database option settings

Database Ownership

Responsibility for managing an individual database rests with the Database Owner. The Database Owner can:

- Add user names for the database
- Grant permission for those users for reading, writing and deleting data
- Perform other tasks in the database as authorized by the System Administrator

Dropping Databases

The owner of a database can remove the database from SQL Server. Removing (or dropping) a database does the following:

- Deletes the database and all the objects in it
- Frees the space allocated for the database
- Deletes references to the database from the system tables in the *master* database

Only the Database Owner can drop a database, and only from the *master* database. You cannot drop a database that is open for reading or writing by any user.

If tables in a database are referenced from other databases, you must remove these references before you can drop the database.

Creating, Altering, and Dropping Tables

The **create table** statement has many options that you can use to define a table in a SQL Server database. When creating a table, some of the information you might specify includes:

- A name for the table

- A name for the columns in the table
- A datatype for each column
- A default value for each column if data is not entered (including whether the column can hold a NULL value
- Any integrity constraints for columns in the table
- Any other information that affects the columns, data rows, and indexes for that table

By default, you create a new table in the currently open database. Table names must be unique for each user. However, since table names can be qualified by database name and user name, different users can create tables of the same name.

You can create a table in a database other than the current database by qualifying the table name with the name of the other database. However, you must be an authorized user of the database in which the table is being created, and you must have **create table** permission in that database.

Table Indexes

A table index is similar in concept to the index of a book. It is a sorted list that points directly to the location of data, just as an index entry in a book points the reader to a particular page. The advantage of an index is that SQL Server does not have to scan an entire table to find a specific data item, just as an index in a book enables a reader to find specific information without having to scan the entire book.

In general, an index speeds up data retrieval by letting SQL Server directly access specific rows in a table. Without an index, SQL Server scans each row, a lengthy process when querying large tables.

Tables may have more than one index. SQL Server decides whether or not to use the indexes for each query submitted for that table. Whenever the data in a table changes, SQL Server automatically changes the table’s indexes to reflect those modifications.

Dropping Tables

When you drop a table, SQL Server removes the specified table, its contents, and all the indexes and privileges associated with it.

You must own a table to be able to drop it. However, no one can drop a table while it is in use—being read or written by a user or a client application.

SQL Server Datatypes

The datatype of a column specifies what kind of data—characters, numbers, or dates—the column holds. For example, the integer (*int*) datatype is used to hold whole numbers.

SQL Server datatypes fall into the following categories:

- Exact number (integer)
- Exact number (decimal)
- Approximate numeric (float, double precision, real)
- Money
- Date / time
- Character (including text)
- Binary (including image)
- Bit

SQL Server and Open Client support their own sets of datatypes. In most cases, there is a direct correspondence between a server datatype and an Open Client datatype. Open Client datatypes are discussed in “Datatypes” on page 6-7.

NULL Values

For each column in a table, you can specify whether or not to allow the NULL value. A NULL value is not the same as zero or blank. Defining columns to allow the NULL value provides the option of leaving that column empty; data may be added at a later time. For example, a NULL entry in the *price* column of the *titles* table does not mean that the book is being given away free, but rather that the price information has not been entered for some reason.

User-Defined Datatypes

SQL Server allows you to name and design your own datatype based on one of the system datatypes. When you define a datatype, you give it a name and associate the following attributes with it:

- The length of the data field
- The datatype's precision and scale, if applicable
- A default value (including NULL)
- The IDENTITY property
- Rules

After defining a datatype, you can use it for any column in the database. For example, *tid* is a user-defined datatype in the *pubs2* sample database that allows up to 6 characters and does not accept the NULL value. For columns in several tables in the *pubs2* sample database, *tid* is used in the definition of other datatypes.

If you add user-defined datatypes in the *model* database, they can be used, by name, in more than one database. When added to *model*, user-defined datatype definitions are known to all new databases.

Datatype Conversion

SQL Server automatically handles many conversions from one datatype to another. Transact-SQL functions allow you to explicitly request other conversions not handled automatically.

For example, SQL Server automatically converts *char* expressions to *datetime* for the purposes of comparison, but you must explicitly request conversion of *char* to *int*. Similarly, you must explicitly convert integer data if you want SQL Server to treat it as character data.

You cannot convert between some datatypes because of inherent incompatibilities between them.

Key Columns

The primary key is the column or combination of columns that uniquely identifies a row. A primary key must contain data; it cannot be NULL. Also, data in every row of a primary key column must be unique.

For example, in the *pubs2* database, the *title_id* column is the primary key of the *titles* table. Each row in the table has a unique value in its *title_id* column that exactly identifies that row.

A foreign key is a column or combination of columns whose values exactly match the primary key, usually in another table. A foreign key need not be unique within its table.

IDENTITY Columns

To ensure that each row of a table is uniquely identifiable, you can add an IDENTITY column to the table when you create it.

A SQL Server IDENTITY column holds a value that is generated by SQL Server and uniquely identifies each row in that table. The IDENTITY column can be used to store sequential numbers—such as invoice numbers, employee numbers, or record numbers—that are guaranteed to be unique.

The first time you insert a row into a table, SQL Server assigns the IDENTITY column a value of 1. For each new row, the IDENTITY value is incremented by one.

Altering Table Structure

You can change a table's structure as well as rename a table or a column within a table.

You can modify a table's structure by:

- Adding columns
- Specifying default values for columns
- Specifying legal values for columns
- Specifying that the column can accept NULL values
- Adding an IDENTITY column
- Dropping default values or specifications for legal values
- Specifying a primary key or a foreign key
- Specifying an index, how to store the index, and where to locate the index

Views

You can create a view that is derived from rows or columns (or both) of one or more tables. The tables from which a view is derived are called its base or underlying tables. A view can also be derived from other views. The definition of a view is stored in the database.

To a user, a view looks exactly like a database table. Users can be given permissions to operate on data available in a view that may be the same or different from permissions granted for that same user on the underlying tables.

Using views provides the following benefits:

- Allows users to focus on specific data

  When you create a view, you can leave out irrelevant data.

- Provides users with simpler data manipulation

  You can predefine complex conditions and qualifications so that users do not have to each time they perform operations on that data.

- Allows different users to see the same data in different ways, even when they are using the same data at the same time

  This ability is particularly important when users with different information needs and skill levels share the same database.

- Provides better security for SQL Server databases

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

Data Integrity

Data integrity refers to the correctness and completeness of data within a database. To enforce data integrity, you can constrain or restrict the data values that users can insert, delete, or update in the database.

For example, the integrity of data in the *pubs2* database requires that a row in the *titles* table must have a publisher in the *publishers* table. You cannot insert books into *titles* that do not have a valid publisher, because doing so violates the data integrity of *pubs2*.

Types of Data Integrity

Listed below are the types of data integrity SQL Server can enforce.

Integrity Type	Description
Requirement	Requires that a column contain a valid value in every row; the column does not allow NULL values.
Check or Validity	Limits or restricts the data values inserted into a table column.
Uniqueness	Requires that no two table rows have the same non-NULL value for a column or combination of columns.

Integrity Type	Description
Referential	Requires that data inserted into a column must already have matching data in the column of another table or another column in the same table.

Default Values

A default is a value that SQL Server inserts into a column if the user does not explicitly enter a value. For example, if most of your customers live in New York, you can specify "New York" as the default value for the State field on an order form. When an order is processed, the user can simply leave the default value in the field or change it if the customer lives in another state.

In a relational database management system, every data element must contain some value—even if that value is NULL. When you assign a default value to a column, SQL Server automatically inserts that value whenever a user does not enter information for the column.

Rules

A rule specifies what values users may or may not enter in a specific column or in any column with a given user-defined datatype. For example, you can create a rule to ensure that users enter all the digits of a product code.

You can connect a rule to a specific column, to several columns, or to a specified user-defined datatype. Whenever a user enters a value using a Transact-SQL **insert** or **update** statement, SQL Server checks the value against the most recent rule that has been bound to the specified column.

To use SQL Server Manager to enable a rule:

1. Create the rule.
2. Bind the rule to a column or user-defined datatype.

The *pub_idrule* rule in the *pubs2* database is an a example of a simple rule. It constrains the values in the *pub_id* column to either one of four specific values or to values beginning with the digits 99 optionally followed by one or two digits each between 0 and 9.

```
create rule pub_idrule
as @pub_id
in ("1389", "0736", "0877", "1622", "1756")
or @pub_id like "99[0-9][0-9]""
```

To apply a rule to a particular column in a particular table, use the **sp_bindrule** procedure and to remove its applicability to a specific column use the **sp_unbindrule** procedure.

To remove a rule from a database use the **drop rule** command.

Triggers

A trigger is a stored procedure that takes effect when you insert, delete, or update data in a specified table. Triggers help maintain consistency among logically related data in different tables.

Triggers are automatic; they work no matter what the source of the data modification, from a user's manual data entry or an application's action. A trigger is specific to one or more of the data modification operations: **update**, **insert**, or **delete**. The trigger is executed once per Transact-SQL statement; it fires immediately after the data modification statements are completed.

The trigger and the statement that fires it are treated as a single transaction that can be rolled back from within the trigger. If a severe error is detected, the entire transaction is cancelled.

Triggers are most useful in the following situations:

- To cascade changes through related tables in the database

  For example, a **delete** trigger on the *title_id* column of the *titles* table could cause a corresponding deletion of matching rows in other tables.

- To disallow, or roll back, changes that violate referential integrity

  For example, you could create an **insert** trigger on *titleauthor.title_id* that rolls back an attempted insert if the inserted value does not match some value in *titles.title_id*.

- To enforce more complex restrictions than rules allow

  For example, a trigger can roll back updates that attempt to increase a book's price by more than 1 percent of the advance.

- To perform simple "what if" analyses

For example, a trigger can compare the state of a table before and after a data modification, and take actions based on that comparison.

Getting More Information

The table below lists sources of information for topics discussed in this chapter.

Topic	**Source(s) of Information**
SQL Server Manager	• *SQL Server Manager User's Guide*
Data integrity	• *Transact-SQL User's Guide*
Database and object names	• *Transact-SQL User's Guide*
Database options	• *SQL Server System Administration Guide*
Datatype conversion	• *SQL Server Reference Manual* • *Transact-SQL User's Guide*
Indexes	• *SQL Server Reference Manual* • *Transact-SQL User's Guide*
Modifying tables and other database objects	• *Transact-SQL User's Guide*
pubs2 sample database	• *SQL Server System Administration Guide* • *Transact-SQL User's Guide*
Views	• *SQL Server Reference Manual* • *Transact-SQL User's Guide*

4 Backing Up and Restoring Data

Introduction

This chapter provides an overview of backup and recovery procedures, including:

- Using the transaction log
- Recovering databases automatically on startup
- Dumping and loading databases
- The role of Backup Server in backup and recovery
- Creating a backup plan

Tracking Database Changes

Changes made to data stored in SQL Server are called transactions. A transaction consists of one or more Transact-SQL statements that succeed or fail as a unit. A transaction is also defined as SQL Server's unit of work.

Each database has its own transaction log. The transaction log automatically records every transaction issued by each user of the database. You cannot turn off transaction logging.

If any statement in a transaction fails to complete, SQL Server reverses all changes made by the transaction.

Automatic Database Recovery

Each time you restart SQL Server after manually shutting it down or after an abnormal termination, SQL Server automatically performs a set of recovery procedures on each database.

The recovery mechanism compares each database to its transaction log. If the log record for a specific change is more recent than the data page, the change is applied to the database from the transaction log. If a transaction was in process and not committed at the time of the failure, the recovery mechanism reverses, or rolls back, all changes that were made by the transaction, ensuring that the entire transaction succeeds or fails as a unit.

Dumping and Loading Databases

In case of media failure, such as a disk crash or a power failure, you can restore your databases only if you have been making regular backups of the databases and their transaction logs. Full recovery depends on regularly backing up databases and database objects.

Database Consistency

Before backing up a database, you should check the integrity of the internal structures of a database. The System Administrator or Database Owner should run database consistency checks on a regular basis.

The Database Consistency Checker (**dbcc**) is a set of utilities for checking the logical and physical consistency of a database. Use the **dbcc** commands:

- Before backing up a database
- As part of regular database maintenance to detect, and often correct, errors before they affect the user's ability to use SQL Server
- If you suspect that a database is damaged

  For example, if using a specific table generates the message "Table corrupt," you can use **dbcc** to determine if other tables in the database are also damaged.

Routine Database and Transaction Log Dumps

Dumping a database makes a copy of the entire database, including both the data and the transaction log.

You can make routine backups of the transaction log, similar to the incremental backups provided by many operating systems. This operation copies the transaction log, providing a record of any database changes made since the last database or transaction log dump.

Dumping a transaction log takes less time and storage space than a full database backup and it should be done more often. Users can keep making changes to the database while the transaction log dump is taking place.

Transaction logs can become full quickly in development environments or in production environments where there is intensive updating of database records. When transaction logs become full, operations of SQL Server stop and user queries in process are aborted. It is important to monitor transaction log size and to activate thresholds that warn when transaction logs are becoming full. You can back up a transaction log and then remove committed transactions to make more space.

Restoring an Entire Database

To restore an entire database after database corruption has occurred, initiate the recovery by loading the most recent database dump.

After you load the database, load each transaction log dump in the order in which it was made. This process reconstructs the database by re-executing the changes recorded in the transaction logs.

When the entire sequence of transaction log dumps has been loaded, the database reflects all transactions that had been committed at the time of the last transaction log dump.

Backup Server

Backup Server is an Open Server program that backs up and restores SQL Server databases. Backup Server can run on the same computer as SQL Server or you can perform backups over the network, using one Backup Server on a remote computer and another on the local computer.

Backup Server supports a type of dumping referred to as dump striping. This allows you to use up to 32 backup devices in parallel, thus splitting the database into approximately equal portions and backing up each portion to a separate device.

While dumps and loads execute, SQL Server and Backup Server use remote procedure calls to exchange instructions and status messages. Backup Server, not SQL Server, performs all data transfer for the dump and load commands. Using Backup Server requires some setup procedures so that SQL Server and Backup Server can communicate across the network.

Creating a Backup Plan

You should develop and test backup procedures for recovering from data corruption. Determine a reasonable backup schedule and adhere to it. If you develop, document, and test your backup procedures ahead of time, you are better prepared to get databases back online in the event of a system crash or media failure.

User Databases

One of the major activities in developing a backup plan is to determine how often to back up your databases. The frequency of backups determines how much work is lost in the event of a media failure. The following guidelines can help you decide when to dump user databases and transaction logs:

- Dump each user database just after you create it, thus providing a base point.
- At a minimum, back up the transaction log daily and the user databases weekly.
- Back up frequently updated databases often (even as much as every half hour), determining the frequency by deciding how much data you can afford to lose in the event of media failure.
- Back up interdependent databases—databases where there are cross-database transactions, triggers, or referential integrity—at the same time, during a period when there is no cross-database data modification activity.
- In addition to routine dumps, dump a database each time you perform any of the following activities:
  - Creating a new index
  - Performing an unlogged operation
  - Dumping the database with a truncated transaction log or no transaction log

System Databases

Follow the guidelines below when backing up SQL Server system databases:

- Back up the *master* database after each command that affects disks, storage, databases, or segments (for example, creating a user database, mirroring a disk, or resetting a configuration variable).
- For further protection, save the scripts that initialize, create or modify databases and make a hard copy of your *sysdatabases*,

sysusages, and *sysdevices* system tables each time you issue one of these commands.

- Each time you change the *model* database, make a new backup.
- If you change permissions on some system procedures, or create your own system procedures, back up the *sybsystemprocs* system database.

You need to restore the *master* database if it is damaged by a media failure or by internal corruption in the database. A damaged *master* database makes itself known in one or more of these ways:

- SQL Server cannot start
- There are frequent or debilitating crashes or input / output errors
- **dbcc** (the database consistency checker) reports damage during a regularly scheduled check of databases

Other situations may require you to restore other system databases, such as *model*.

Getting More Information

The chart below lists sources of information for topics discussed in this chapter.

Topic	Source(s) of Information
Backup Server	• *SQL Server System Administration Guide*
Database consistency	• *SQL Server System Administration Guide*
Dumping and loading	• *Configuring Sybase SQL Server for your platform* • *SQL Server System Administration Guide*
Error messages	• *SQL Server Reference Supplement* • *Error Messages Guide*
Global variables	• *SQL Server System Administration Guide*
SQL Server Manager	• *SQL Server Manager User's Guide*
System stored procedures	• *SQL Server System Administration Guide*

5 Accessing Data with Transact-SQL

Introduction

This chapter describes Transact-SQL, Sybase's enhanced version of Structured Query Language (SQL) for accessing and manipulating data in your SQL Server databases. The chapter discusses:

- The basic Transact-SQL statements
- Accessing a SQL Server database
- Selecting data from one or more tables
- Adding, deleting, or changing data
- Using queries within other queries (subqueries)
- Using built-in functions in queries
- Accessing data row by row with cursors
- Handling Transact-SQL errors
- Controlling the flow of an application
- Using stored procedures

Basic Transact-SQL Statements

Transact-SQL provides statements that are used for different purposes:

- A query is a request for the retrieval of data, using the **select** statement. For example:

  ```
  select au_lname, city, state
  from authors
  where state = 'CA'
  ```

  This statement retrieves authors' last names and city and state of residence from the *authors* table for those authors who live in California.

  After processing a query, Transact-SQL returns a result set. If no data meets the criteria specified in the query, Transact-SQL returns a message to that effect.

- Data modification refers to an addition, deletion, or change to data, using the **insert**, **delete**, or **update** statement, respectively. For example:

```
insert into authors (au_lname, au_fname, au_id)
values ("Smith", "Gabriella", "999-03-2346")
```

This statement adds Gabriella Smith's name and ID number to the *authors* table.

- Other SQL statements are instructions to perform data definition operations, for example:

  ```
  drop table authors
  ```

  This statement removes the *authors* table from the *pubs2* database.

Each SQL statement begins with a keyword, such as **insert**, that names the basic operation being performed. Many SQL statements also have one or more keyword phrases, or clauses, that tailor the statement to a specific need.

Accessing a SQL Server Database

When your SQL Server account was created, you were probably assigned a default database. When you log in, SQL Server connects you to your default database unless you explicitly specify otherwise. If you were not assigned a default database, you are connected to the *master* database when you log in.

Sybase provides an easy-to-use Windows query tool, **wisql**, with which to query SQL Server databases. When you double-click on the **wisql** icon in the Sybase program group, the **wisql** entry window allows you to:

- Connect to a database
- Enter queries
- Display rows, columns, and messages returned by SQL Server

Use **wisql** as an administrative and management query tool or as a testing tool during development of client applications for SQL Server.

To access a database other than your default database, enter:

```
use database_name
```

in the **wisql** entry window. For example, to open the *pubs2* database, enter the SQL command:

```
use pubs2
```

If you are not known in the database, the owner of the database must give you access to it.

Selecting Data from a Table

The **select** statement retrieves data from a SQL Server database. A simple **select** statement is:

```
select au_fname, au_lname
from authors
where city = "Oakland"
```

This statement returns the first and last names of all authors in the *authors* table who live in Oakland.

Specifying Columns in a Query

The list that follows the **select** keyword, called the select list, specifies the columns from which to retrieve data. The select list can contain:

- A series of one or more column names separated by commas
- An asterisk (*) to represent all columns in the table
- One or more expressions, separated by commas

  An expression can be a constant, column name, function, subquery, or any combination of these elements.

Renaming Columns in Query Results

You can give a column a more readable name in the results of a query. For example, the following statement:

```
select Publisher = pub_name, pub_id
from publishers
```

returns the *pub_name* column title as “Publisher” in the query results.

Computations on Retrieved Data

You can perform computations with data from numeric columns in a select list. For example, to see what a projected sales increase of 25 percent for all the books in the *titles* table looks like, enter the following statement:

```
select title_id, total_sales, total_sales * 1.25
from titles
```

Distinct Values

The optional **distinct** keyword eliminates duplicate rows from the results of a **select** statement. For example, suppose you want to retrieve all the author identification codes in the *titleauthor* table.

Without the **distinct** keyword, SQL Server returns a row for each occurrence of each value. To return each code only once, use the following statement:

```
select distinct au_id
from titleauthor
```

Clauses for the *select* Statement

Clauses in a **select** statement narrow the scope of the data that returns. Listed below are some commonly used clauses.

Clause	Description
from	Lists all the tables and views containing columns included in the select list and in the **where** clause.
where	Specifies the criteria for which rows are retrieved.
group by	Divides a result set into groups; frequently used to compute an aggregate value for each group (for example, total sales for each region).
having	Specifies criteria for the **group by** clause, similar to the effect of the **where** clause on the **select** statement (for example, sales for each region having more than a certain dollar amount).
order	Sorts the results by column name or number or by the value of an expression.
compute	Used with the **group by** clause to generate summary values (for example, to return each region's total sales as an additional row in the query results).

Retrieving Data from Several Tables

The join operation makes it possible to retrieve data from several tables in a single result set. The ability to join data from multiple tables enables the building of queries independent of the basic table structure of the database.

To join two tables, you must include a joining condition in the **where** clause of the query. The joining condition stipulates which columns from the two tables must correspond and the basis under which they must correspond (equality, inequality, and so on). The most common type of join is based on equality.

For example, in the statement:

```
select title, price
from titles, publishers
where pub_name = "Binnet and Hardley" and
publishers.pub_id = titles.pub_id
```

the joining condition:

```
publishers.pub_id = titles.pub_id
```

indicates that the query should return only those rows in which the value of the *pub_id* column in the *publishers* table is the same as the value of the *pub_id* column in the *titles* table.

If you do not use a joining condition in a query that accesses multiple tables, the results will be the cross-product of the query, which consists of all possible combinations of the qualifying rows from all the tables listed in the **from** clause. If we alter the previous example by removing its joining condition so that it reads:

```
select title, price
from titles, publishers
where pub_name = "Binnet and Hardley"
```

the query would return the cross-product, which consists of all the rows from the *publishers* table in which the *pub_name* value is Binnet & Hardley and all the rows in the titles table, since there would be no qualification based on data in the *titles* table. The cross-product is rarely the intended result.

The columns joined in the joining condition do not have to have the same name or be of the same datatype, but they must be of datatypes that can be compared with one another and reference the same domain of values.

As in the example above, a joining condition can coexist in the **where** clause with other conditions that restrict the rows returned by the query.

Adding, Changing, and Deleting Data

To maintain up-to-date information in SQL Server tables, users need to modify the data, delete some data and add new data. Listed below are the Transact-SQL statements for modifying data.

Statement	Description
insert	Adds new rows to a table
update	Changes existing rows in a table
delete	Removes rows from a table

Constraints

You cannot execute a Transact-SQL data modification statement that would violate a constraint defined on a table or group of tables. This restriction includes column-level constraints, table-level constraints, and referential integrity constraints, which involve inter-table relationships.

Adding New Data

Use the **insert** statement in conjunction with the following clauses to add rows to a table:

- Use the **values** clause to specify values for some or all of the columns in a new row.

  For example, the following statement adds a new row to the *publishers* table, giving a value for every column in the row:

  ```
  insert into publishers
  values ("1622", "Jardin, Inc.", "Camden", "NJ")
  ```

- Use a **select** clause in an **insert** statement to add rows using values from other tables.

  For example, suppose the table *newtitles* contains rows of title information in the same format as *titles*. The following statement adds books published after 1994:

  ```
  insert into newtitles
  select *
  from titles
  where datepart(year, pubdate) > 1994
  ```

Specifying Columns to Receive Values

You also can specify the columns for which you want to insert values.

For example, the following query specifies adding data to the *pub_id* and *pub_name* columns of the publishers table:

```
insert into publishers (pub_id, pub_name)
values ("1756", "The Health Center")
```

When you specify values for only some of the columns in a row, SQL Server takes one of the following actions for the columns with no values:

- Generates a unique, sequential value if the column has the IDENTITY property
- Enters a default value if one exists for the non-specified column or user-defined datatype
- Enters NULL if it was allowed for the column and no default value exists for the column or user-defined datatype
- Returns an error message and does not add the row if none of the above conditions exists

In the example above, SQL Server enters the NULL value in the *city* and *state* columns because no value was given for these columns in the **insert** statement and NULL values are allowed in these columns. The results of the query

```
select *
from publishers
where pub_id = "1756"
```

produces the following results:

```
pub_id    pub_name           city    state
-------   -----------------  ------  -------
1756      The Health Center  NULL    NULL
```

Changing Existing Data

Use the **update** statement to change one or more rows in a table. Follow the **update** keyword with the name of the table or view. As in all the data modification statements, you can change the data in only one table at a time. However, you can change more than one column value in a single statement.

The **update** statement specifies:

- The table to change.
- The row or rows to change (in the **where** clause).
- The column or columns to change (in the **set** clause).
- The new data (in the **set** clause).

For example, if Mary McMartin changes her last name to McMartin-McNab, the following statement updates the *authors* table accordingly:

```
update authors
set au_lname = "McMartin-McNab"
where au_lname = "McMartin"
```

If Mary changed her address when she changed her name, you could handle both column changes in the same statement, if the affected columns are in the same table:

```
update authors
set au_lname = "McMartin-McNab",
address = "22 Main Street"
where au_lname = "McMartin"
```

Deleting Data

Use the **delete** statement to delete rows from tables. For example, to remove the row for Jardin, Inc., from the *publishers* table, enter the following statement:

```
delete publishers
where pub_name = "Jardin, Inc."
```

The **where** clause specifies which rows to remove. If you do not include this clause in the **delete** statement, all rows in the table are removed.

Subqueries

Queries that are nested inside of other queries are called subqueries. Subqueries are used to return one or more values for comparison in a **where** or **having** clause in an outer query or data modification statement.

For example, to find all the books in the *titles* table that have the same price as *Straight Talk About Computers*, you could find the price of the book and then use a second query to find all books with this price.

However, you can use a subquery, as follows, to retrieve all the information with a single query:

```
select title, price
from titles
where price =
   (select price
    from titles
    where title = "Straight Talk About Computers")
```

Transact-SQL Built-In Functions

Transact-SQL provides built-in functions as extensions to SQL for performing various tasks. You can use built-in functions anywhere in a Transact-SQL statement where an expression is allowed.

SQL Server's built-in functions are categorized as follows:

- System functions, most of which return information from the system tables
- String functions for manipulating *character* and *binary* values
- Text functions for manipulating *text* and *image* values
- Mathematical functions for trigonometry, geometry, and other number handling
- Date functions, for manipulating *date* values
- Datatype conversion functions for converting expressions from one datatype to another and for formatting dates

Accessing Data Row by Row with Cursors

A cursor is a variable that is associated with a query. Cursors are useful for browsing applications that retrieve an unknown amount of data in an outer query, examining that data row by row, and either running an inner query or running an **update** or **delete** statement on the outer row. Cursors allow very complex manipulation of data on a row by row basis.

Cursor Position

After its associated query has been executed, the cursor has one of the following positions:

- Before the first row of the query's results set
- After the last row of the query's results set
- On a row in the query's results set

If the cursor was declared as updatable, a user can select a set of data using a cursor, browse through it row by row, and modify or delete the row to which the cursor is currently pointing.

Transact-SQL Statements Involving Cursors

Transact-SQL statements involving cursors are listed below:

Statement	Description
declare cursor	Associates a cursor name with a query
open	Executes the query on the server
close	Invalidates the cursor's name and frees the cursor's resources
fetch	Gets the rows returned by the query for the client
update	Updates the row the cursor currently points to
delete	Deletes the row the cursor currently points to

Establishing the Scope of a Cursor

The scope of a cursor determines the region in which the cursor is known. When a cursor's scope no longer exists, its name is no longer valid.

The regions that SQL Server uses to define cursor scopes are listed below.

Region	Starting Point	Ending Point
Session	A client logs onto the server.	The client logs off.
Stored procedure	A stored procedure begins execution.	The procedure completes execution.
Trigger	A trigger begins execution.	The trigger completes execution.

Viewing, Updating, and Deleting Rows Using Cursors

Cursors can be either updatable or read-only. If a cursor is read-only, you can only fetch the data; you cannot update or delete. If you do not plan to update or delete rows through a cursor, declare it as read-only by using the **read only** keyword in the **declare cursor** statement, because there are restrictions on the contructs allowed in an updatable cursor that do not apply to a read-only cursor. By default, SQL Server assumes that a cursor is updatable.

You can specify that only certain columns accessed by the query are updatable by using the **for update** keywords in the **declare cursor** statement. For example, the following statement means that the user can update values in the *city* and *state* columns retrieved by the *pubs_crsr* cursor:

```
declare pubs_crsr cursor
for select pub_name, city, state
from publishers
for update of city, state
```

User-Defined Transactions

Transact-SQL lets you group a set of SQL statements into a user-defined transaction. In the absence of a user-defined transaction, each individual statement is considered its own transaction and is committed independently.

For example, you might have a set of statements that update related tables. You would not want any of the tables updated unless they all were changed; therefore, you would like to define the set of **update** statements into a single transaction that can be rolled back if one of the updates fails.

User-defined transactions are used for data modification statements, **insert**, **update** and **delete**, that you want to be able to commit to the database or undo as a unit.

Transaction Management Statements

Statements for managing transactions are listed below:

Statement	Description
begin transaction **commit transaction**	These two statements mark the beginning and end of a transaction block; all statements between these statements are included as part of the transaction (unless a **rollback transaction** statement is executed).
rollback transaction	Undoes the transaction, either back to its beginning, or back to a savepoint.
save transaction	Defines a savepoint inside a transaction; when a transaction reaches a savepoint, all work up to that point is committed and cannot be rolled back.

Any user can define a transaction; no permission is required for any of the transaction statements.

Transact-SQL Errors

Transact-SQL provides various error handling techniques, including:

- Capturing return status from stored procedures
- Defining customized return values from stored procedures
- Passing parameters from a procedure to its caller
- Getting reports from global variables such as *@@error*
- Using the **raiserror** or **print** statement to direct error messages to the screen (you can localize these messages for different languages)
- Using **set** statement options to customize message format, show processing statistics, and provide other diagnostic aids for debugging your Transact-SQL statements

Controlling the Flow of an Application

Transact-SQL provides keywords, called control-of-flow language, that let you control the flow of execution of statements. You can use these keywords in single statements, stored procedures, and triggers.

Control-of-flow language lets you refine and control the operation of an application. Transact-SQL's control-of-flow language transforms standard SQL into a very high-level programming language.

The control-of-flow keywords are listed below:

Keyword	Function
if	Defines conditional execution
...else	Defines alternate execution when the if condition is false
begin	Begins a statement block
...end	Ends a statement block
while	Repeats execution of statements while condition is true
break	Exits from the end of the next outermost while loop
...continue	Restarts a **while** loop
declare	Declares local variables
goto *label*	Goes to the position in the statement block specified by *label*
return	Exits unconditionally
waitfor	Sets delay for statement execution
print	Prints a user-defined message or local variable on the user's screen
raiserror	Prints a user-defined message or local variable on the user's screen and sets a system flag in the global variable *@@error*
/* *comment* */	Inserts a comment anywhere in a SQL statement

Stored Procedures

Stored procedures are collections of SQL statements and control-of-flow language. SQL Server prepares and saves an execution plan when it runs a stored procedure, so subsequent execution is very fast.

Stored procedures can:

- Take parameters
- Call other procedures

- Return a status value to a calling procedure or batch to indicate that the procedure succeeded or failed
- Return values of parameters to a calling procedure or batch
- Be executed on remote servers

The ability to write stored procedures greatly enhances the power, efficiency, and flexibility of Transact-SQL. Stored procedures greatly improve the Transact-SQL performance. In addition, you can execute stored procedures on other SQL Servers if both your SQL Server and the remote SQL Server are set up to allow remote logins.

Creating and Executing Stored Procedures

Create stored procedures with the **create procedure** statement.

Execute a stored procedure with the **execute** statement. You can also use the name of the stored procedure alone, as long as it is the first word of a statement or batch.

Stored Procedures and Permissions

Stored procedures can serve as security mechanisms because you can grant a user permission to execute a stored procedure without granting the user any other permissions on tables or views referenced in the procedure. To do this, the owner of the stored procedure must be the owner of the tables and views referenced in the procedure.

Getting More Information

The table below lists sources of information for topics discussed in this chapter.

Topic	Source(s) of Information
All Transact-SQL Topics	• *Transact-SQL User's Guide* • *SQL Server Reference Manual*
Keywords	• *SQL Server Reference Supplement*
System Procedures	• *SQL Server System Administration Guide* • *SQL Server Reference Manual*
Stored Procedures	• *SQL Server Reference Manual*

6 Building Client / Server Applications

Introduction

SQL Server for Workplace UNIX includes products that make it easy for C programmers to develop customized client applications to access SQL Server. This chapter discusses these Sybase connectivity products, including:

- A description of Client-Library and Net-Library, Open Client components included with SQL Server for Workplace UNIX
- A discussion of how to create Client-Library applications
- An overview of how Open Client establishes connections between clients and servers
- A description of DB-Library, Sybase's set of application programming interfaces provided for compatibility with release 4.x applications

SQL Server for Workplace UNIX also includes 16-bit and 32-bit ODBC Driver Kits for accessing SQL Server data from a variety of Windows or Windows NT front-end applications.

What Is Open Client?

Open Client is a programming application interface (API) used to develop customized client applications that access SQL Server. The Open Client libraries manage all communications between a client application or tool and SQL Server databases.

In addition, if you purchase the Sybase Open Server product, you can develop applications that allow any data source, information application, or system service to respond to Open Client requests as if it were SQL Server.

The Open Client connectivity products:

- Shield you from having to implement the specifics of networks, operating systems, transport protocols, and data access methods
- Let you write system-independent client applications

  Using Open Client for client application development ensures that when you port an application to another operating system, the application can provide the same functions, whether or not the underlying operating system supports them.

The SQL Server installation includes the following Open Client products:

- Client-Library, a set of library functions that takes maximum advantage of SQL Server features; this API allows you to develop your own client applications that access data in SQL Server.
- CS-Library, a set of utility routines used by both Open Client and Open Server applications.
- Net-Library, a set of network drivers that transparently supports various network protocols for connection between client applications and SQL Server.
- DB-Library, a set of APIs provided for compatibility for applications developed using previous releases of SQL Server.

These products offer a choice of interfaces to the Sybase architecture for use with third- or fourth-generation programming languages, including development tools from Sybase or other vendors.

Client-Library

Client-Library includes routines that you include in a C-language client application to:

- Send commands to SQL Server
- Process the results of those commands
- Set application properties
- Handle error conditions
- Provide information about an application's interaction with SQL Server

A Client-Library program is compiled and run in the same way as any other C program.

Asynchronous Processing

Client-Library uses asynchronous processing, so that applications can make server requests while continuing to process other work. Asynchronous processing provides the following capabilities:

- You can structure an application to do other useful work while waiting for results from a network or external device or while waiting for completion of a server process.

- Users can cancel an operation midstream.
- An application can yield control to other applications while SQL Server is preparing results, thus eliminating idle time waiting for complex query processing.

Developing Generic Applications

Client-Library is a generic interface. As such, Client-Library does not enforce or reflect any restrictions of SQL Server.

When writing a Client-Library application, keep in mind the application's ultimate target server (for example, SQL Server or an Open Server application). If you are unsure about what is legal for a particular server application, consult its documentation.

Client-Library Features

Client-Library provides the following features for developing client / server applications:

- Cursor commands for stepping through a result set one record at a time
- Calls for sending Transact-SQL stored procedures and remote procedure calls
- Open Client user-defined datatypes
- Message and error handling through callback routines
- Inline error and message handling
- Dynamic SQL statements
- Regular row, compute row, stored procedure return parameters, and return status
- International localization at various levels

Implementing a Client-Library Application

The following steps describe the basic procedure required for implementing an Open Client / C application.

1. Set up the Client-Library programming environment by:
   - Initializing Client-Library
   - Allocating a context structure
   - Setting CS-Library properties for the context

   CS-Library is the Open Client common library; it contains utilities used by both Open Client and Open Server. At least two calls to CS-Library are required for all Client-Library applications.
2. Define error handling.

   Most applications use callback routines to handle Client-Library and server error and informational messages.
3. Connect to SQL Server by:
   - Allocating a connection structure
   - Setting its properties
   - Opening a connection to the server
   - Setting options for the connection
4. Send a command to SQL Server by:
   - Allocating a command structure
   - Initiating a command
   - Defining any parameters that the command requires
   - Sending the command
5. Process the results of the command by:
   - setting up the result set for processing
   - getting information about the result set and a result items
   - binding the result item to program data space

   Result rows can then be accessed and processed. Processing for cursor results includes several steps specific to handling of cursors.
6. Complete the application by:
   - Deallocating command structure space
   - Closing the connection to the server
   - Leaving Client-Library
   - Deallocating context structure space

Control Structures

To send commands to SQL Server, a Client-Library application allocates three types of control structure, as shown below.

Structure	Function
CS_CONTEXT	Defines a specific application “context” or operating environment
CS_CONNECTION	Defines a specific client / server connection
CS_COMMAND	Defines a “command space” in which commands are sent to a server

Through these structures, a client application sets up its environment, connects to servers, sends commands, and processes results.

Client/Server Connections

Clients connect to SQL Server or Open Server applications across a network. To connect with a server, the client application must:

1. Determine the name of the server.
2. Find out where on the network that server is located.

The Server Name

Use Client-Library routines to connect to the server. There are several ways to determine the name of the server with which to connect. You may:

- Hard-code the name of the server in the call
- Set the server name to a program variable

  This method provides more flexibility and is particularly useful for applications that connect to several servers.
- Omit the server name from the connection call, indicating that the server name is to take its value at run time

  The server name is then determined by the run-time value of the DSQUERY environment variable or, if this variable is not set, the server name defaults to “SYBASE.”

Making the Connection

After Open Client learns which server to connect to, it must figure out how to make the connection. Open Client obtains this information from the network configuration files, which contain network address information for clients and servers. These files consist of entries for all servers to which a client might connect as well as Net-Library driver information. As such, they serve as the "address book" for the client.

The files used to specify client and server addresses and network drivers vary somewhat by platform. Refer to the platform-specific *Open Client/Server Supplement* for complete information.

Result Set Processing

After an Open Client application sends a command to a server, it processes any results generated by the command or by the server. Listed below are the results that Client-Library routine calls can return.

Result Type	Description
Regular row results	Generated when SQL Server executes a Transact-SQL **select** statement; contains zero or more rows of data.
Cursor row results	Generated when an application executes a Client-Library cursor open command; contains zero or more rows of data.
Parameter results	Contains a single row of parameters (for example, message parameters or stored procedure return parameters).
Stored procedure return status results	Generated by the execution of a stored procedure; consists of a single row containing a return status.
Compute row results	Generated by the execution of a Transact-SQL **select** statement that contains a **compute** clause; consists of a single row containing a number of columns equal to the number of row aggregates in the **compute** clause.
Message results	Contains an *id*, which an application can retrieve. If parameters are associated with a message, they are returned as a separate parameter result set following the message result set.
Describe results	Indicates the existence of descriptive information returned as the result of a Dynamic SQL statement.

Result Type	Description
Format results	Makes advance format information available to an application that needs format information before the result set is processed (for example, a gateway application that repackages SQL Server results before sending them to a foreign client).

A single command can generate more than one type of result. For example, a Transact-SQL statement that executes a stored procedure can generate multiple regular row and compute row result sets, a parameter result set, and a return status result set. For this reason, it is important that applications be coded to handle all results that SQL Server can generate.

The simplest way for an application to process all possible results is by using a loop construct. Inside the loop, a set of switch statements implements processing of whatever type of result is currently available for processing.

Datatypes

Client-Library supports a wide range of datatypes. A header file provided with Open Client contains type definitions for all of the Open Client datatypes. Using type definitions hides variations in datatype implementations on different platforms, making Open Client programs more portable.

A client application declaring program variables uses Open Client type definitions in its declaration section. For example:

```
CS_CHAR      buffer[40];
CS_INT       result_type, count;
CS_MONEY     profit;
```

In most cases, Open Client datatypes correspond directly to SQL Server datatypes.

Callbacks

Callbacks are user-supplied routines that are automatically called by Client-Library whenever certain triggering events, known as callback events, occur. For example, a client message callback event occurs when Client-Library generates an error message. When Client-Library recognizes a callback event, it automatically calls the appropriate callback routine.

Listed below are the types of callbacks:

Type of Callback	When Called
Client Message	In response to a Client-Library error or informational message
Completion	When an asynchronous Client-Library routine completes
Encryption	During the connection process, in response to a SQL Server request for an encrypted password
Negotiation	During the connection process: • In response to a SQL Server request for login security labels • In response to a SQL Server challenge
Server Message	In response to a SQL Server error or informational message.
Signal	In response to an operating-system SIGIO or SIGPOLL signal

Error and Message Handling

Both SQL Server and Client-Library generate messages in response to a wide range of error and informational conditions. Each error message has a number, text, and severity level.

An application can handle Client-Library and SQL Server messages using one of two methods:

- Inline message handling

  Inline error handling has the advantage of being under an application's direct control, allowing the application to check for messages at times that it specifies

- Callbacks

  Advantages of callbacks are:

  - They are relatively automatic. After they are installed, callbacks are triggered whenever a message occurs.
  - They centralize message-handling code.
  - They provide a way for an application to gracefully handle unexpected errors.

Most applications use callbacks to handle messages, but an application that is running on a platform and language combination that does not support callbacks must use the inline method.

Building a Client-Library Executable

To compile a Client-Library/C application, use a Sybase-certified C compiler for your platform. Compile and link your application as specified by your compiler and operating-system documentation.

All Client-Library applications require the header file provided with Open Client that contains type definitions and declarations required by Client-Library routines.

Localizing Client-Library Applications

You can localize an application to run in a specific national language environment. An application that is localized typically:

- Generates messages in a local language and character set
- Uses local date and time formats

The localization information of an application is described by a locale name. A locale name is a character string that represents a language/character set/sort order combination. For example, the locale name "fr" might represent the combination french/iso_1/binary. Sybase provides some pre-defined locale names, and you can define others, as well.

Character Set Conversion Between SQL Server and Client-Library

When a localized Client-Library application connects to SQL Server, the server checks to see if it supports the application's language, character set, and sort order. If it does, then SQL Server handles the translation and issues messages in the client's language and character set. If SQL Server does not support the application's language, character set, or sort order, it issues a warning and does not make the connection.

Client-Library Sample Programs

Open Client Client-Library/C includes sample programs that illustrate how to use the Client-Library routines. These programs access the *pubs2* sample database, which is also provided as part of your SQL Server for Workplace UNIX installation.

Net-Library

Net-Library furnishes a standard interface to most of the available network protocol implementations and provides transport protocol and operating system independence.

Because data can reside on any hardware platform supported by Sybase and can be accessed with common communications protocols, Net-Library's capabilities enable application development and deployment independent of the network. By specifying a Net-Library at run time, you can choose network protocols as needed.

DB-Library

DB-Library is the programming interface that Sybase provided with earlier releases. SQL Server for Workplace UNIX includes DB-Library for compatibility with existing applications.

Using DB-Library with Client-Library

Although you can make calls to both DB-Library and Client-Library in a single application, you must treat the libraries as separate within the application. That is, you cannot use functions in one library with the structures, datatypes, commands, or result sets from the other library.

You can convert DB-Library programs to their Client-Library equivalents.

Getting More Information

The table below lists sources of information for topics discussed in this chapter.

Topic	Source(s) of Information
Callbacks	• *Open Client Client-Library/C Reference Manual*
Client-Library API	• *Open Client Client-Library/C Reference Manual* • *Open Client Client-Library/C Programmer's Guide*
Connection routines	• *Open Client Client-Library/C Reference Manual*

Topic	Source(s) of Information
Control structures	• *Open Client Client-Library/C Reference Manual* • *Open Client Client-Library/C Programmer's Guide*
Client-Library Datatypes	• *Open Client Client-Library/C Reference Manual* • *Open Client and Open Server Common Libraries Reference Manual*
DB-Library API	• *Open Client DB-Library/C Reference Manual*
Error handling	• *Open Client Client-Library/C Reference Manual*
Localization	• *Open Client Client-Library/C Reference Manual*
Message handling	• *Open Client Client-Library/C Reference Manual*
Net-Library	• *Open Client/Server Supplement* for the relevant platform
Results Processing	• *Open Client Client-Library/C Programmer's Guide*

Glossary

aggregate function

A function that works on a set of cells to produce a single answer or set of answers, one for each subset of cells.

alias

A name that allows a SQL Server user to be known in a database by another name.

argument

A value supplied to a function or procedure that is required to evaluate the function.

arithmetic expression

An expression that contains only numeric operands and returns a single numeric value. In Transact-SQL, the operands can be of any numeric datatype. They can be functions, variables, parameters, or they can be other arithmetic expressions. Synonymous with numeric expression.

automatic recovery

A process that runs every time SQL Server is stopped and restarted. The process ensures that all transactions that have completed before SQL 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.

base tables

The tables on which a view is based. Also called "underlying" tables.

batch

One or more Transact-SQL statements terminated by an end-of-batch signal, which submits them to the SQL Server for processing.

binding

An association between a default or rule and a database table column or a user-defined datatype.

browse mode

A method that Client-Library applications can use to browse through database rows, updating their values one row at a time. Cursors provide similar functionality and are generally more portable and flexible.

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 They can be used in Transact-SQL statements.

bulk copy

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

callback

A routine that Open Client calls in response to a triggering event, known as a callback event.

callback event

In Open Client, an occurrence that triggers a callback routine.

capabilities

A feature of a client/server connection that determine the types of client requests and server responses permitted for that connection.

cascading delete

A delete operation implemented with a trigger that deletes data from a table based on a deletion from another table. A cascading delete is often performed to delete detail data when master data is deleted in a master/detail application.

character expression

An expression that returns a single character-type value; can include literals, concatenation operators, functions, and column identifiers.

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.

check constraint

A limit on the values users can insert into a column of a table. A **check** constraint specifies a search condition, which must not evaluate to FALSE for the value being inserted through an **insert**, **update**, or **bcp**.

client

In client/server systems, the part of the system that sends requests to servers and processes the results of those requests.

Client-Library

Part of Open Client; a collection of routines for use in writing client applications. Client-Library is alibrary designed to accommodate cursors and other advanced Sybase features.

column

A column contains an individual value within a row. Columns are ordered and named.

column-level constraint

Limits the acceptable values for a specified column.

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.

connection structure

A hidden Client-Library structure that defines a client/server connection within a context.

constant expression

An expression that always returns the same value. For example, “3 + 5” is a constant expression.

context structure

A CS-Library hidden structure that defines an application “context,” or operating environment, within a Client-Library application.

control-of-flow language

Programming-like constructs (such as **if**, **else**, **while**, **goto**, **label:**) provided by Transact-SQL to enable the user to control the flow of execution of Transact-SQL statements.

conversion

See **character set conversion**.

current row

The row to which a cursor points. A fetch against a cursor retrieves the current row.

cursor

A symbolic name associated with a Transact-SQL **select** statement through a declaration statement. A cursor behaves much like a file pointer to a series of file records, where the cursor acts as a pointer to the query results.

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

In SQL Server, 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. The database device can be any piece of disk or a file in the file system that is used to store databases and database objects.

database object

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 or who has been assigned ownership by the creator of a database. 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 holds and how the data is stored. System datatypes include *char*, *int*, *money*, and so on. You can construct your own datatypes in SQL Server based on the SQL Server system datatypes.

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.**

DB-Library

Part of Open Client; a collection of routines for use in writing client applications.

default

1. The database object that specifies a value that SQL Server inserts for a column when no value is provided. 2. The option used when you do not explicitly specify an option.

default database

The database accessed by default when you log into SQL Server.

default language

1. The language that displays that user’s prompts and messages, as set with the **sp_modifylogin** system procedure or the **language** option of the **set** command. 2. The language that SQL Server uses to display prompts and messages users unless you chooses a different language.

delete

The deletion of one or more rows from a database table.

device

See **database device**.

device I/O

Reads and writes performed by SQL Server to a database device.

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.

dump

1. A backup copy of a database or a transaction log. 2. The process of creating a backup.

dump striping

Interleaving of dump data across several **dump volumes**.

Dynamic SQL

A Client-Library feature that allows an application to execute SQL statements containing variables whose values are determined at run time.

expression

A computation, column data, built-in function, or subquery that can be evaluated.

foreign key

A key column in a table that logically depends on a primary key column in another table; a column (or combination of columns) whose values are required to match a primary key in some other table or be NULL.

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**.

gateway

An application that acts as an intermediary for clients and servers that cannot communicate directly. Acting as both client and server, a gateway application passes requests from a client to a server and returns results from the server to the client.

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.

identifier

A string of characters used to identify a database object, such as a table name or column name.

IDENTITY column

The column in a table that contains a system-generated value that uniquely identifies each row within that table.

index

A storage structure that speeds up data retrieval by pointing SQL Server to where a table column’s data is stored on disk.

insert

The addition of a row to a database table.

int

A signed 32-bit integer value.

join

A basic operation in a relational database management system. A join links the rows in two or more tables by comparing the values in specified columns.

key

A column used to identify a row, often used as the index column for a table.

keyword

A word or phrase that is reserved for exclusive use by Transact-SQL. Also known as a reserved word.

load

1. A copy of a database, used to recover from a media failure. 2. The process of copying a database backup to recover a database.

locale

The geographical environment in which a program is running. The locale determines the language, sort order, and date and time formatting conventions.

locale name

A character string that represents a language and character set pair. Locale names are listed in the locales file. SQL Server predefines some locale names, but you can define additional locale names and add them to the locales file.

localization

The process of setting up an application to run in a specific native language environment. An application that is localized typically generates messages in a local language and character set and uses local datetime formats.

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*.

***master* database**

The system database that controls the user databases and the operation of SQL Server as a whole. The *master* database keeps track of such information as user accounts, ongoing processes, and system error messages.

memory allocation

The allocation of memory to SQL Server; a SQL Server configuration option.

mirror

See **disk mirror**.

***model* database**

A template for new user databases. Each time the **create database** command is issued, SQL Server makes a copy of *model*.

multibyte character set

A character set that includes characters encoded using more than one byte. EUCJIS and Shift-JIS are examples of character sets that include several types of characters represented by multiple bytes in a Japanese language environment.

null

Having no explicitly assigned value. NULL is not equivalent to zero, or to blank.

objects

See **database object**.

Open Client

A client application that communicates requests to SQL Server.

operating system

A group of programs that translates your commands to the computer, helping you perform such activities as creating files, running programs, and printing documents.

Operator

A SQL Server user in charge of performing server-wide database operations such as backing up and restoring databases.

outer query

The principal query in a statement containing a subquery.

parameter

1. An argument to a stored procedure. 2. A value passed between routines.

permission

The authority to perform a specified action on a certain object, such as executing a stored procedure, or running a particular command.

precision

The maximum number of digits that can be represented in a *decimal*, *numeric*, or *float* column.

primary key

The column or columns whose values uniquely identify a row in a table.

privilege

See **permission**.

procedure

See **stored procedure**.

qualified

A database object whose name is preceded by the name of the database and the object owner.

query

1. A request for the retrieval of data. 2. Any SQL statement that manipulates data.

recovery

The process of rebuilding one or more databases from database dumps and log dumps. See also **automatic recovery**.

referential integrity

The rules governing data consistency, specifically the relationships among the primary keys and foreign keys of different tables. SQL Server addresses referential integrity with user-defined triggers.

referential integrity constraint

A constraint requiring that data inserted into a "referencing" table that defines the constraint must have matching values in a "referenced" table.

remote procedure call

A mechanism for executing a stored procedure on a remote SQL Server.

role

A SQL Server user's authorization level: System Administrator, System Security Officer, or Operator. A user may be assigned more than one role, and more than one user may have the same role.

rollback transaction

A Transact-SQL statement used within a **user-defined transaction** (before a **commit transaction** has been received) to undo any changes that were made to the database.

row

A set of related **columns** that describes a specific entity in a table or view. Rows are unordered.

rule

A SQL Server specification that controls the domain of acceptable values for a table column or a user-defined datatype.

sa

See **System Administrator**.

savepoint

A marker inside a **user-defined transaction** that specifies a point to which you can roll back (cancel) a transaction. When you roll back the transaction to the savepoint, all statements before the savepoint are committed, and all statements after the savepoint are cancelled.

scale

The number of digits to the right of the decimal point in a *numeric* or *decimal* datatype. The scale of a datatype cannot be greater than its **precision**.

segment

A label that points to one or more database devices that are available to a specific database. You use segments to control the placement of tables and indexes on specific database devices.

select list

The list of data to be retrieved by a **select** statement.

server

In client/server systems, the part of the system that processes client requests and returns results to clients.

severity level number

The severity of an error condition. Errors with severity levels of 19 and above are fatal errors.

shared memory

Area of SQL Server memory that holds information required to coordinate multiple operating system processes and make them appear to clients as a single SQL Server process.

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.

statement

An instruction given to SQL Server.

statement block

A series of Transact-SQL statements enclosed between the keywords **begin** and **end;** the statements are treated as a unit in control-of-flow constructs such as **if...else** and **while**.

stored procedure

A collection of preprocessed Transact-SQL statements stored under a name in SQL Server. SQL Server-supplied stored procedures are called **system procedures**.

subquery

A **select** statement that is nested inside another **select**, **insert**, **update**, or **delete** statement, or inside another subquery.

System Administrator

The user in charge of administrative activities unrelated to specific applications. These activities include managing disk storage, granting create database permission to SQL Server users, and running diagnostic and repair functions.

system databases

The databases that SQL Server automatically installs—the master database (*master*), which controls user databases and the operation of the 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 information 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 as mechanisms for accomplishing database administration and other activities that involve updating system tables.

System Security Officer

The user in charge of security-sensitive activities on SQL Server, such as creating, dropping, and locking user accounts, and changing passwords of other users. See also **role**.

system table

A SQL Server data dictionary table. The system tables keep track of information about the SQL Server as a whole and about each user database.

table

A database object used to store data. Each row in a table describes one occurrence of an entity, and each column describes one characteristic of the entity.

table-level constraint

A constraint that limits values on more than one column of a table.

thresholds

Space usage values of database and log segments. When free space in a segment falls below a threshold value, a stored procedure executes.

Transact-SQL

An enhanced version of Structured Query Language (SQL). Applications use Transact-SQL to communicate with SQL Server.

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. The trigger action is the action for which the trigger is specified, and the trigger conditions are the conditions that cause the trigger to take effect.

type conversion function

A function used to convert expressions of one datatype into another datatype, whenever these conversions are not performed automatically by SQL Server.

update

An addition, deletion, or change to data, involving the **insert**, **delete**, **truncate table**, or **update** statements.

update lock

A lock granted to an updating process while a table is being searched for qualifying rows. When the actual update is performed, the qualifying rows are locked exclusively.

user-defined datatype

A definition of the type of data a column can contain, created by the user and defined in terms of the existing system datatypes. Rules and defaults can be bound to user-defined datatypes (but not to system datatypes).

user-defined transaction

A block of Transact-SQL statements delimited by a **begin transaction** and **commit transaction** statement, allowing users to group any number of simple transactions as a single unit that can be completed or canceled.

variable

A named piece of memory, used to store a value.

view

A virtual table consisting of selected columns and rows from one or more tables or other views.

Index