Student Workbook for *Murach's SQL for SQL Server*

Mike Murach & Associates, Inc.
2560 West Shaw Lane, Suite 101
Fresno, CA 93711-2765
(559) 440-9071 • (800) 221-5528
murachbooks@murach.com • www.murach.com

How to use this workbook

This workbook is designed as a companion to our book, *Murach's SQL for SQL Server*. If you are using this workbook as part of a college course or corporate training program, your instructor may give you specific information about how to use the material it contains. In general, though, you can use this material to determine if you understand and can apply the information presented in each chapter of the book. The topics that follow present some additional information about how to use each of the components of this workbook.

Objectives

To help you prepare for taking the tests and doing the exercises your instructor assigns, this workbook includes objectives for each chapter. These objectives describe what you should be able to do after reading the chapter. If you want to, you can review the objectives for a chapter before you start reading it so you know what material to focus on. Keep in mind, however, that you may not fully understand the objectives at that point. In any case, you'll definitely want to review the objectives once you've finished the chapter.

If you study the objectives, you can see that the first objectives for each chapter from chapter 2 on are what we refer to as *applied objectives*. These ask you to apply what you've learned as you code SQL statements. If you can do the exercises in this workbook, you have met these objectives.

After the applied objectives for each chapter, you'll find what we refer to as *knowledge objectives*. (Because chapter 1 doesn't teach coding skills, all the objectives are knowledge objectives.) These objectives define skills like identifying, describing, and explaining the required concepts, terms, and procedures. In general, you should be able to do the knowledge objectives, even if you have trouble with the applied objectives. If you are familiar with all the terms presented in the chapter and you can answer the self-study questions, you have met these objectives.

Summaries

This workbook also includes a summary of the information presented in each chapter. You can use this summary to review the main concepts and coding techniques presented in the chapter after you finish reading the chapter. Then, if you aren't clear about any of these concepts or techniques, you can review the topics that present them. That will help you to prepare for the chapter test and the exercises your instructor may assign.

Terms

In addition to the summary, a list of terms is presented for each chapter. After reading a chapter, you should scan the list to check your comprehension of each term. Then, if you don't have a clear understanding of a term, you'll want to locate it within the chapter to see how it's used. As you do that, keep in mind that you're not expected to write definitions of the terms. You just need to understand how they're used in the context of coding SQL statements.

Self-study questions

The self-study questions for each chapter test your knowledge of the objectives for that chapter. To prepare for taking the test for a chapter, then, you should be sure that you can answer the self-study questions. If you can't answer a question, you'll want to study the related material in the chapter.

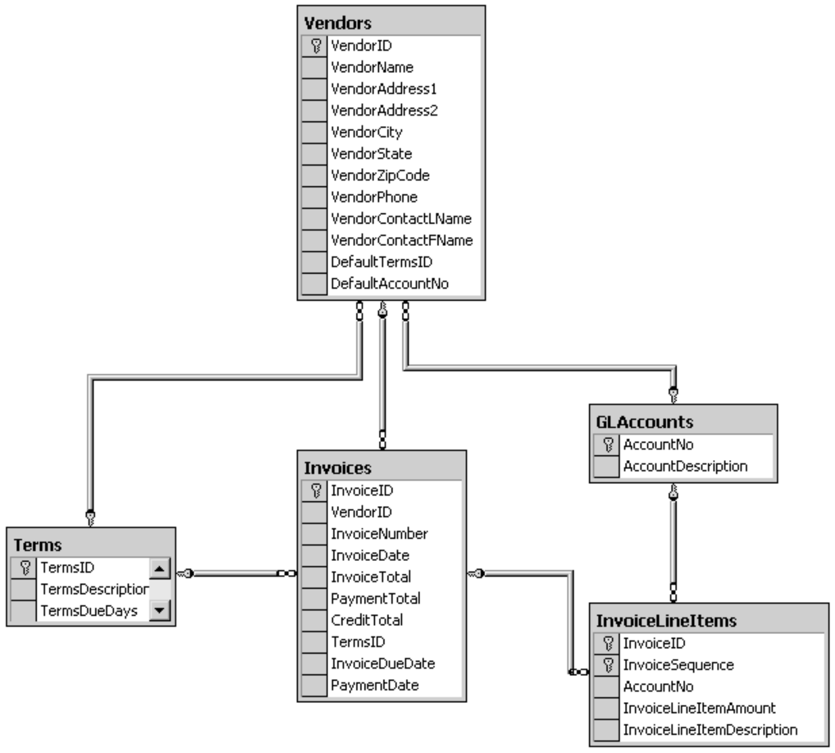
Exercises

To help you practice the skills you'll learn in the book, this workbook provides exercises for each chapter except chapter 1, which doesn't present any coding techniques. Each exercise has you practice the coding skills you learned in the corresponding chapter. Typically, your instructor will assign one or more of the exercises for each chapter. If not, you can do the exercises on your own to practice and solidify your coding skills.

If you're working on your own PC, you'll need to set up your system before you can do these exercises. Specifically, you'll need to install SQL Server, you'll need to install the databases and SQL files used by the exercises on your C drive, and you'll need to attach the databases to the server. The procedures you'll use to do these tasks are described in appendix A of the book, and the files you need are on the book's CD.

Most of the exercises are based on the AP (accounts payable) database that's used throughout the book. The only exceptions are those exercises that have you create a new database and the last exercise in chapter 8, which has you use a database named Examples that contains several small tables used in some of the book examples.

The purpose of the AP database is to track vendors and their invoices for the payables department of a small business. It consists of the five tables shown in the diagram that follows.



The top-level table is the Vendors table, which contains one row for each of the vendors the company purchases from. Its primary key is VendorID, which is an identity column, so SQL Server automatically generates its value whenever a new vendor is created.

Each invoice that's received from a vendor is stored as a row in the Invoices table. Here again, the primary key for this table, InvoiceID, is an identity column. This table also includes two foreign keys: VendorID, to relate each invoice to a vendor in the Vendors table, and TermsID, to relate each invoice to a row in the Terms table.

The InvoiceLineItems table contains one row for each line item of each invoice. Its primary key is a combination of InvoiceID and InvoiceSequence. InvoiceID relates each line item to an invoice, and InvoiceSequence gives each line item a unique primary key value.

The Terms table records payment terms for invoices, such as "Net Due 10 Days" (the invoice must be paid in 10 days). The primary key of this table is TermsID, which is an identity column. Each invoice has a TermsID column to identify the payment terms for that invoice. And each Vendor has a DefaultTermsID column that provides the default terms for new invoices from that vendor.

Finally, the GLAccounts table provides general-ledger account information that the company uses to track its expenses. For example, one account might be used for advertising expenses, while another might be for office supplies. The primary key of this table is AccountNo. Then, each line item includes an AccountNo column that specifies which account the purchase should be charged to, and each Vendor has a DefaulAccountNo column that provides a default account number for new invoices.

Chapter 1
An introduction to relational databases and SQL

Objectives

- Identify the three main hardware components and the two main software components of a client/server system.
- Describe how a database table is organized, and explain the relationship between a column, a row, and a cell.
- Describe how the tables in a relational database are related, and identify the three types of relationships that can exist between two tables.
- Describe how the columns in a table are defined.
- In general terms, explain the advantages of the relational database model over the hierarchical database model, the network database model, and conventional file systems.
- Describe the relationship between standard SQL and the various commercial dialects of SQL. Identify the dialect of SQL used on Microsoft SQL Server.
- Describe the difference between DML statements and DDL statements.
- Explain the difference between an action query and a SELECT query.
- Describe coding techniques that can make your SQL code easier to read, debug, and maintain.
- Describe the proper use of comments.

Summary

- A multi-user system typically consists of *clients* and *servers* connected by a *network*.
- The data used by a *client/server system* is typically stored in *relational databases* that are managed by a *database management system* (*DBMS*).
- *Application software* running on the client communicates with the DBMS by sending *SQL queries* through the *data access API*. Then, the DBMS processes the query and returns any *query results*.
- An *application server* can be used to store *business components* that do part of the processing of an application, such as processing database requests.
- *Web servers* can be used to store *web applications*, which receive requests from a *web browser* running on a client. A web application can perform part of its processing using *web services*, which are also stored on a web server.
- The data in a relational database is stored in *tables* that consist of *rows* (*records*) and *columns* (*fields*). Each table typically has a *primary key* that uniquely identifies each row in the table.

- The tables in a relational database are related to each other through their *primary keys* and *foreign keys*. Most relationships are *one-to-many*.
- Each column in a table is defined with a *data type* that determines the type of information it can hold. Each column definition also indicates whether the column can contain *null values* and whether it has a *default value*.
- Each table can have an *identity column* whose value is generated automatically by the DBMS when a row is added to the table.
- In comparison to earlier data models, relational databases require more system resources. But they're easier and more flexible to use because less programming is required to handle their data.
- *SQL* is the standard language that's used to handle the data in relational databases. Database manufacturers add *extensions* to the standards to create their own SQL *dialects*. The SQL dialect that's used by Microsoft SQL Server is called *Transact-SQL*.
- The *data manipulation language (DML)* consists of the SQL statements that programmers use to work with the data in a database.
- The *data definition language (DDL)* consists of the statements used to work with database objects, like tables. It's usually used by *database administrators* (*DBAs*).
- To retrieve data from a *base table*, you use a SQL SELECT statement that stores the rows and columns you request and the *calculated values* you specify in a *result table*, or *result set*. You can also use a SELECT statement to *join* the data from two or more tables in a single result set.
- To add rows to a table, you use the SQL INSERT statement. To update rows in a table, you use the SQL UPDATE statement. And to delete rows from a table, you use the SQL DELETE statement. These are known as *action queries*.
- A *view* is a database object that contains a SELECT statement. Views can be used to restrict the data that a user is allowed to access or to present data in a form that's easy for the user to understand.
- A *stored procedure* is a database object that contains one or more precompiled SQL statements. A *trigger* is a special type of stored procedure that's executed automatically as the result of an action query. And a *user-defined function* is a special type of procedure that can return a single value or an entire table.

Terms

client
server
database server
network
client/server system
local area network (LAN)
enterprise system
wide area network (WAN)
network operating system
database management system (DBMS)
back-end processing
back end
application software
data access API (application programming interface)
front-end processing
front end
SQL (Structured Query Language)
SQL query
query results
application server
web server
business component
web application
web service
web browser
thin client
relational database
table
row
column
record
field
cell
primary key
composite primary key
non-primary key
unique key
index
foreign key
one-to-many relationship
one-to-one relationship
many-to-many relationship
data type
null value
default value
identity column
hierarchical database
parent/child relationship
network database
SEQUEL (Structured English Query Language)
Oracle
SQL/DS (SQL/Data System)
DB2 (Database 2)
ANSI (American National Standards Institute)
levels of compliance
levels of conformance
core specification
package
SQL dialect
extension
SQL variant
Transact-SQL
open source system
data manipulation language (DML)
data definition language (DDL)
database administrator (DBA)
base table
result table
result set
calculated value
query
join
inner join
outer join
cross join
action query
comment
block comment
single-line comment
view
virtual table
viewed table
stored procedure
input parameter
output parameter
control-of-flow language
trigger
user-defined function (UDF)

Self-study questions

1. What are the hardware components in a typical client/server system? What is the function of each?
2. What are the software components in a typical client/server database application? What is the function of each?
3. How is a table in a relational database organized? What are its basic parts?
4. How are the tables in a relational database related to each other? What are the three types of relationships that can exist between tables?
5. What basic information is used to define a column in a table?
6. What's an identity column?
7. How is the default value of a column used?
8. What advantages does the relational database model offer over earlier database systems?
9. What is standard SQL? How does it differ from commercial dialects of SQL?
10. What dialect of SQL is used on Microsoft SQL Server?
11. What are the SQL DML statements used for? the DDL statements? Which statements are you most likely to use as an application programmer?
12. What SQL statement do you use to retrieve rows from a table? to add rows to a table? to remove rows? to change data values?
13. What's the difference between a SELECT query and an action query?
14. What are some coding techniques you can use to make your SQL code easier to read, debug, and maintain?
15. What two types of comments can you include in your SQL code? How is each one typically used?

Chapter 2
How to work with a SQL Server database

Objectives

- Using the Enterprise Manager, view the database diagram for a database and the definitions of the tables in the database.
- Given a SQL statement, use the Query Analyzer to enter and execute that statement.
- Use Books Online to look up information about SQL and SQL Server.
- Identify the two major components of SQL Server, and briefly describe the function of each.
- Identify the two ways that SQL Server can authenticate a login ID.
- Briefly describe the function of each of these client tools: the Enterprise Manager; the Query Analyzer; and Books Online.
- Differentiate between the Query Analyzer and the Query Designer.
- Describe how an application program gains access to a SQL Server database, and identify three common data access models.

Summary

- Microsoft SQL Server consists of a *database server* that provides database management services and *client tools* that let you work with a database and its data.
- The *Enterprise Manager* provides a visual interface that lets you work with database objects. You can use it to view the relationships between the tables in a database, the definitions of the columns in a table, and other table properties.
- You use the *Query Analyzer* to enter, execute, and save queries. Its *Object Browser* makes it easy to check the names of tables and columns and to view the data in a table.
- The *Query Designer* builds queries based on selections you make, so you don't have to enter the SQL statements yourself. It works best for simple queries where the efficiency of the resulting code isn't an issue.
- *Books Online* is a tool that lets you look up information in a complete set of SQL Server reference manuals.
- An application uses a *data access model* to connect to and work with SQL Server databases. Three common models are *ADO.NET* (for Visual Basic .NET applications), *JDBC* (for Java), and *ADO* (for Access and Visual Basic 6).

Terms

database server
database engine
client tools
MSDE (Microsoft SQL Server Desktop Engine)
SQL Server Service Manager
Enterprise Manager
console tree
node
database diagram
check constraint
Query Analyzer
Object Browser
Query Designer
Books Online
data access model
ADO (ActiveX Data Objects)
JDBC (Java Database Connectivity)
ADO.NET
driver
data table
dataset
data adapter
data command
data connection
disconnected data
disconnected data architecture

Self-study questions

1. What are the two major components of SQL Server? What's the function of each one?
2. What does the Enterprise Manager let you do?
3. What's a database diagram? How do you display one?
4. How do you display the definitions of the columns in a table?
5. What are two methods that SQL Server can use to authenticate a login ID? How do they differ?
6. What's the function of the Query Analyzer?
7. How do you execute a query from the Query Analyzer? What are the Grids and Messages tabs of its Query window used for?
8. If an error occurs when a query is executed, what's displayed in the Query window? What are some common causes of SQL errors?
9. What's the Object Browser? How would you use it?
10. What's the difference between the Query Analyzer and the Query Designer?
11. When and how would you use Books Online?
12. In general terms, how does an application program gain access to a SQL Server database?
13. What are three common data access models? Name at least one programming language that each one is commonly used with.

Exercise 2-1 Use the Enterprise Manager to view the AP database

In this exercise, you'll use the Enterprise Manager to view the definition of the AP database. Some of the steps refer to figures in the book so that you can use them as guides for your work.

Start the Enterprise Manager and locate the AP database

1. Use the Windows Start menu to locate and start the Enterprise Manager.
2. When the Enterprise Manager workspace is displayed, expand the Microsoft SQL Servers node, the SQL Server Group node, and the node for the instance of SQL Server that contains the AP database. (If the database engine isn't currently running, this will start it.) At this point, you should see folders for all of the services provided by SQL Server.
3. Expand the Database node to display a list of the databases that the server manages. Then, expand the node for the AP database to display a list of the types of objects that can be stored in the database as shown in the left part of the window in figure 2-3 of the text.

View the relationships between the tables in the database

4. Select the Diagrams node to display the database diagrams for the AP database in the right side of the window. In this case, the database contains a single diagram.
5. Double-click on the diagram to display it. This diagram should look like the one shown in figure 2-4.
6. Review the information that's contained in each table, note the primary key of each table, and try to identify the relationships between the tables.
7. When you're done, close the window to return to the Enterprise Manager workspace.

View the table definitions

8. Select the Tables node to display the tables in the AP database. Then, locate the Vendors table, right-click on it, and select the Design Table command. A window like the one in figure 2-5 should be displayed (you may have to use the View menu to set the view to Detail to match the display in the figure).
9. Review the definition of each column in this table. In particular, note the data type of each column, whether it allows null values, and whether it has a default value. Also note that the VendorID column is defined as an identity column.
10. Click on the Table and Index Properties button in the toolbar to display the Tables tab of the Properties dialog box. Note the owner of the database. (If you're using MSDE, it should be dbo.) Also notice that this dialog box indicates the identity column for the table.
11. Click on the Relationships tab to display the foreign keys for the table. Then, drop down the Selected relationship combo box to see that this table has three foreign keys. Notice that the names of these keys identify the table on the other side of the relationship. Select each of these foreign keys and note the columns in the primary key and foreign key table that form the relationship.

12. Click on the Indexes/Keys tab. Then, use the combo box at the top of the dialog box to display the settings for each key and index.
13. Click on the Check Constraints tab and note that no check constraints are defined for this table.
14. When you're done reviewing the properties for the Vendors table, close the Properties dialog box and then close the Design Table Window.
15. Repeat this procedure for the other tables in the AP database. Then, close the Enterprise Manager.

Exercise 2-2 Query the AP database

In this exercise, you'll use the Query Analyzer to enter and execute a SELECT statement against the AP database. Then, you'll save the query, open and execute another query, and correct any errors you encounter.

Enter, execute, and save a query

1. Start the Query Analyzer and connect to the database server as shown in figure 2-7 of the text. If you encounter problems, make sure that SQL Server is running and that you're using the proper authentication.
2. When the Query Analyzer window is displayed, select the AP database from the Select Database combo box in the toolbar.
3. Enter the query that follows, being careful to type each column name correctly:

   ```
   Select InvoiceNumber, InvoiceDate, InvoiceTotal,
       PaymentTotal, CreditTotal,
       InvoiceTotal - PaymentTotal – CreditTotal As BalanceDue
   From Invoices
   Where InvoiceTotal – PaymentTotal – CreditTotal > 0
   Order By InvoiceDate
   ```

   Use continuation lines and indentation as appropriate (your choices may differ from what's shown above). Note how the Query Analyzer displays the keywords in blue as you enter them.
4. Press F5 to execute the query and display the results in the Grids tab of the results pane. If errors are detected or the results don't look like those in figure 2-8, correct the problem before you continue.
5. Click the Save button in the toolbar to display the Save Query dialog box. Then, navigate to the C:\Murach\SQL for SQL Server\Scripts\Chapter 02 folder on your computer. Enter SelectUnpaidInvoices for the file name, and then click on the Save button to save the statement.

Open another query and correct errors

6. Click on the New Query toolbar button to open another Query window, and select the master database from the Select Database combo box. Then, use the Open command in the File menu to display the Open Query File dialog box, and open the query named FormatVendorAddress in the C:\Murach\SQL for SQL Server\Scripts\Chapter 02 folder.
7. Click the Parse Query button to see that the syntax of this statement is valid. Then, click the Execute Query toolbar button to execute it. When you do, an error message is displayed in the Messages table of the Results pane indicating that 'Vendors' isn't a valid object name. That's because this table isn't in the master database. To fix this error, select the AP database.
8. Execute the query again. This time, you'll get a message indicating that 'VendorZip' isn't a valid column name. To find the correct column name, open the Object Browser window. Then, expand the AP database, the User Tables folder, the Vendors table, and the Columns folder. Locate the correct column name and modify the SELECT statement accordingly.

9. Execute the query one more time. If it works correctly, the result set will consist of 122 rows, starting with Computer Library in Phoenix, AZ. If it doesn't, correct the problem and execute the query again. Once you have it working, save it and close the Query Analyzer.

Exercise 2-3 Use Books Online

In this exercise, you'll use Books Online to search for and display information in the online manuals.

1. Start Books Online. When you do, the Entire Collection subset should be selected and the Contents tab should be displayed. If not, select this subset and tab.
2. Review the books that are available from the Contents tab and notice that most of them have little or nothing to do with coding SQL. Now, select SQL Programmer: Transact-SQL as the Active Subset to see what books it contains.
3. Expand the Using the SQL Server Tools book. Then, expand the User Interface Reference book and the SQL Query Analyzer Help book. Finally, click on the Overview of SQL Query Analyzer topic to display it in the right pane.
4. Click on the Favorites tab to display it. Then, right-click in this tab and select the Add command to add the Query Analyzer topic.
5. Click on the Contents tab to return to it. Then, click the Next toolbar button to display the next topic in this book.
6. Switch back to the Favorites tab and double-click on the topic you added to display it. Then, remove this topic from the tab by right-clicking on it and selecting Remove.
7. Click on the Index tab. Then, enter "object b" to highlight the Object Browser entry. Double-click on this entry to display a dialog box that lists the available topics. Select the Using Object Browser topic to display it in the right pane.
8. Click on the Querying from Object Browser link in the right pane to display that topic. Then, click on the Locate toolbar button to display the Contents tab with the currently displayed topic highlighted.
9. Click on the Search tab, enter query file directory, and click on the List Topics button. Double-click on the first topic in the list to display it in the right pane. Notice that all occurrences of the words query, file, and directory are highlighted.
10. Click the Back toolbar button one or more times to display previous topics. Then, click the Forward button to display subsequent topics.
11. Continue experimenting with the various features of Books Online until you feel comfortable with this tool. Then, close the program.

Chapter 3
How to retrieve data from a single table

Objectives

- Given a specification for a result set that requires only the language elements presented in this chapter, write the SELECT statement.
- Identify the four basic clauses of a simple one-table SELECT statement and describe what each clause does.
- Identify the five ways you can code the column specification in a SELECT clause.
- Identify the two ways you can assign a column alias to an output column in a result set.
- Describe how the order of precedence for arithmetic operators affects the result of a numeric expression.
- Describe the effect of the ALL and DISTINCT keywords on the result set.
- Describe the use of the comparison and logical operators in a search condition.
- Given a SELECT statement that uses any of the language elements presented in this chapter, explain what each clause does.

Summary

- In a SELECT statement, the SELECT clause identifies which columns should be retrieved from the table in the FROM clause. The WHERE clause specifies a search condition that lets you *filter,* or limit, the rows that are retrieved. And the ORDER BY clause lets you sort the retrieved rows into the sequence you specify.
- To assign a name to a column that's different from its name in the base table, you code a *column alias* using the AS keyword in the SELECT clause.
- An *expression* is a combination of column names and operators that evaluate to a single value. In the SELECT clause, you can code *arithmetic expressions, string expressions,* and expressions that include one or more *functions*.
- The *order of precedence* determines which operations in an arithmetic expression will be done first, second, and so on. You can use parentheses to clarify or override the order of precedence.
- A string expression consists of columns that contain character data and/or literal values that are *concatenated* using the *concatenation operator* (+).
- You can use the DISTINCT keyword in the SELECT clause to eliminate duplicate rows in the result set and the TOP clause to return a subset of selected rows.
- A *Boolean expression* results in a value of True, False, or Unknown. A search condition in the WHERE clause is made up of one or more Boolean expressions, and a row is included in the result set only if all of the conditions are true.

- The search condition in a WHERE clause can use *comparison operators* (like = or <), *logical operators* (like AND or OR), and the IN, BETWEEN, LIKE, and IS NULL operators to specify the conditions a row has to meet to be included in the result set.
- You can sort the rows in a result set by one or more expressions in either ascending or descending sequence.

Terms

keyword	literal value	comparison operator
filter	string literal	logical operator
Boolean expression	string constant	compound condition
predicate	arithmetic expression	subquery
expression	arithmetic operator	string pattern
column alias	order of precedence	mask
substitute name	function	wildcard
string expression	parameter	null value
concatenate	argument	nested sort
concatenation operator	date literal	

Self-study questions

1. What are the four basic clauses of a simple SELECT statement that retrieves data from a single table? What would you code in each one?
2. What are five ways that you can code the column specification in a SELECT statement?
3. What's a column alias? What are two ways you can create one in Transact-SQL?
4. What's a string expression? How do you code one?
5. How does the order of precedence for arithmetic operators affect the result of an arithmetic expression?
6. What's a function? What are the parameters (or arguments) of a function, and how do you supply them?
7. How do the ALL and DISTINCT keywords affect which rows are included in a result set?
8. How does the TOP clause affect which rows are included in a result set?
9. What are comparison operators? What are logical operators? How are they used in search conditions?
10. What's the effect of the IN operator on a search condition? the BETWEEN operator? the LIKE operator? the IS NULL clause?
11. When you include a sort expression in a SELECT statement, what's the default sort order? How do you change that?
12. Name at least one other option for sorting a result set besides using a column name from the base table.

Exercises

1. Write a SELECT statement that returns three columns from the Vendors table: VendorContactFName, VendorContactLName, and VendorName. Sort the result set by last name, then by first name.
2. Write a SELECT statement that returns four columns from the Invoices table, named Number, Total, Credits, and Balance:

Number	Column alias for the InvoiceNumber column
Total	Column alias for the InvoiceTotal column
Credits	Sum of the PaymentTotal and CreditTotal columns
Balance	InvoiceTotal minus the sum of PaymentTotal and CreditTotal

   a. Use the AS keyword to assign column aliases.

   b. Use the = assignment operator to assign column aliases.

3. Write a SELECT statement that returns one column from the Vendors table named Full Name. Create this column from the VendorContactFName and VendorContactLName columns. Format it as follows: last name, comma, first name (for example, "Doe, John"). Sort the result set by last name, then by first name.
4. Write a SELECT statement that returns three columns:

InvoiceTotal	From the Invoices table
10%	10% of the value of InvoiceTotal
Plus 10%	The value of InvoiceTotal plus 10%

   (For example, if InvoiceTotal is 100.0000, 10% is 10.0000, and Plus 10% is 110.0000.) Only return those rows with a balance due greater than 1000. Sort the result set by InvoiceTotal, with the largest invoice first.

5. Modify the solution to exercise 2a to filter for invoices with an InvoiceTotal that's greater than or equal to $500 but less than or equal to $10,000.
6. Modify the solution to exercise 3 to filter for contacts whose last name begins with the letter A, B, C, or E.
7. Write a SELECT statement that determines whether the PaymentDate column of the Invoices table has any invalid values. To be valid, PaymentDate must be a null value if there's a balance due and a non-null value if there's no balance due. Code a compound condition in the WHERE clause that tests for these conditions.

Chapter 4
How to retrieve data from two or more tables

Objectives

- Given a specification for a result set, use the explicit or implicit syntax to code an inner join that returns data from a single table or multiple tables.
- Given a specification for a result set, use the explicit or implicit syntax to code an outer join that returns data from a single table or multiple tables.
- Given a specification for a result set, code a union that combines data from a single table or multiple tables.
- Explain when column names need to be qualified.
- Explain the proper use of correlation names.
- Describe the difference between an inner join and an outer join.
- Describe the differences between a left outer join, a right outer join, a full outer join, and a cross join.
- Given a SELECT statement that uses any of the language elements presented in this chapter, explain how the join or union works.

Summary

- A *join* combines columns from two or more tables into a result set based on the *join condition* you specify. In most cases, the join condition compares key values in the tables, finding all the rows in a foreign key table for a given primary key.
- Since the columns used in a join condition come from different tables, they may have the same column name. In that case, you include the table names with the column names to create *qualified column names*.
- If the table names are long, you can assign a shorter *table alias* or *correlation name* to the table in the query.
- If you join data from tables in different databases or on different servers, you may have to qualify the table names with the name of the server, database, and owner.
- You can use the AND and OR operators to create compound join conditions.
- An *inner join* is the most common type of join. It returns only the rows that satisfy the join condition.
- An *outer join* retrieves all rows that satisfy the join condition, plus unmatched rows in one or both tables. A *left outer join* includes all the rows from the first table. A *right outer join* includes all the rows from the second table. And a *full outer join* includes all the rows from both tables.
- A *cross join* joins each row from the first table with each row from the second table.

- A *self-join* joins a table with itself. It typically includes the DISTINCT keyword so that duplicate rows are eliminated.
- Today, you use the *explicit syntax* for joining tables by coding the JOIN and ON keywords in the FROM clause. Prior to the SQL-92 standards, however, joins were created using the *implicit syntax*, in which the join condition was coded in the WHERE clause.
- A *union* combines the result sets of two or more SELECT statements into one result set. The SELECT statements can query the same or different tables, but they must specify the same number of columns, and the corresponding columns in the result sets must have compatible data types.

Terms

join
join condition
inner join
ad hoc relationship
qualified column name
explicit syntax
correlation name
table alias
fully-qualified object name
partially-qualified object name
self-join
interim result set
interim table
implicit syntax
theta syntax
outer join
left outer join
right outer join
full outer join
cross join
Cartesian product
union

Self-study questions

1. What's a join?
2. When do the column names in a join condition need to be qualified?
3. What's a correlation name? When would you use one?
4. When do you need to qualify the table names in a join condition? What are the parts of a fully-qualified object name?
5. How do you create a compound join condition?
6. What's the difference between an inner join and an outer join?
7. Name three types of outer joins. How do they differ from one another?
8. What's a self-join? a cross join?
9. What's the difference between the implicit and explicit syntax for a join? Which are you most likely to use?
10. What's a union? How do you code one?
11. What rules apply to the columns that you specify in a union?

Exercises

Unless otherwise stated, use the explicit join syntax.

1. Write a SELECT statement that returns all columns from the Vendors table inner-joined with the Invoices table.
2. Write a SELECT statement that returns four columns:

VendorName	From the Vendors table
InvoiceNumber	From the Invoices table
InvoiceDate	From the Invoices table
Balance	InvoiceTotal minus the sum of PaymentTotal and CreditTotal

The result set should have one row for each invoice with a non-zero balance. Sort the result set by VendorName in ascending order.

3. Write a SELECT statement that returns three columns:

VendorName	From the Vendors table
DefaultAccountNo	From the Vendors table
AccountDescription	From the GLAccounts table

The result set should have one row for each vendor, with the account number and account description for that vendor's default account number. Sort the result set by AccountDescription, then by VendorName.

4. Generate the same result set described in exercise 2, but use the implicit join syntax.
5. Write a SELECT statement that returns five columns from three tables, all using column aliases:

Vendor	VendorName column
Date	InvoiceDate column
Number	InvoiceNumber column
#	InvoiceSequence column
LineItem	InvoiceLineItemAmount column

Assign the following correlation names to the tables:

Vend	Vendors table
Inv	Invoices table
LineItem	InvoiceLineItems table

Sort the final result set by Vendor, Date, Number, and #.

6. Write a SELECT statement that returns three columns:

VendorID	From the Vendors table
VendorName	From the Vendors table
Name	A concatenation of VendorContactFName and VendorContactLName, with a space in between

   The result set should have one row for each vendor whose contact has the same first name as another vendor's contact. Sort the final result set by Name.

   Hint: Use a self-join.

7. Write a SELECT statement that returns two columns from the GLAccounts table: AccountNo and AccountDescription. The result set should have one row for each account number that has never been used. Sort the final result set by AccountNo.

   Hint: Use an outer join to the InvoiceLineItems table.

8. Use the UNION operator to generate a result set consisting of two columns from the Vendors table: VendorName and VendorState. If the vendor is in California, the VendorState value should be "CA"; otherwise, the VendorState value should be "Outside CA." Sort the final result set by VendorName.

Chapter 5
How to code summary queries

Objectives

- Given a specification for a result set that requires any of the language elements presented in this chapter, write the SELECT statement.
- Describe the difference between a scalar aggregate and a vector aggregate.
- Given a SELECT statement that uses any of the language elements presented in this chapter, explain how the GROUP BY and HAVING clauses affect the result set.
- Explain the differences between the HAVING clause and the WHERE clause.
- Describe the use of the WITH ROLLUP and WITH CUBE operators.

Summary

- In contrast to a *scalar function*, which operates on a single value and returns a single value, an *aggregate function* operates on a series of selected values and returns a single summary value.
- A query that contains an aggregate function can be called a *summary query*.
- Common aggregate functions are AVG, which figures the average of the values in an expression; SUM, which totals the values in an expression; MIN and MAX, which find the lowest and highest values in an expression; and COUNT(*), which gives a count of the number of rows that are selected by a query.
- You use the GROUP BY clause to group the rows of a result set based on one or more columns or expressions. You typically use it in conjunction with an aggregate function.
- A *scalar aggregate* is an aggregate function that returns a single value for all the rows in the result set. In contrast, a *vector aggregate* returns a value for each group within the result set.
- You can use the HAVING clause to specify a search condition for a group or an aggregate. In contrast to the search condition in the WHERE clause, the HAVING search condition is applied *after* the rows in the result set are grouped or summarized.
- The ROLLUP and CUBE operators are SQL Server extensions that let you add extra summary rows to a result set that uses grouping and aggregates.

Terms

scalar function	column function	scalar aggregate
aggregate function	summary query	vector aggregate

Self-study questions

1. What's an aggregate function?
2. What does the SUM function do? the MAX function? the COUNT(*) function?
3. How does the GROUP BY clause in a SELECT statement affect the result set?
4. What's the difference between a scalar aggregate and a vector aggregate?
5. Why would you code the HAVING clause in a SELECT statement?
6. What's the difference between the HAVING clause and the WHERE clause? Can you code an aggregate function in either one?
7. If a SELECT statement includes the WITH ROLLUP or WITH CUBE operator, how does that affect the result set?

Exercises

1. Write a SELECT statement that returns two columns from the Invoices table: VendorID and PaymentSum, where PaymentSum is the sum of the PaymentTotal column. Group the result set by VendorID.
2. Write a SELECT statement that returns two columns: VendorName and PaymentSum, where PaymentSum is the sum of the PaymentTotal column. Group the result set by VendorName. Return only 10 rows, corresponding to the 10 vendors who've been paid the most.

   Hint: Use the TOP clause and join Vendors to Invoices.
3. Write a SELECT statement that returns three columns: VendorName, InvoiceCount, and InvoiceSum. InvoiceCount is the count of the number of invoices, and InvoiceSum is the sum of the InvoiceTotal column. Group the result set by vendor. Sort the result set so that the vendor with the highest number of invoices appears first.
4. Write a SELECT statement that returns three columns: AccountDescription, LineItemCount, and LineItemSum. LineItemCount is the number of entries in the InvoiceLineItems table that have that AccountNo. LineItemSum is the sum of the InvoiceLineItemAmount column for that AccountNo. Filter the result set to include only those rows with LineItemCount greater than 1. Group the result set by account description, and sort it by descending LineItemCount.

   Hint: Join the GLAccounts table to the InvoiceLineItems table.
5. Modify the solution to exercise 4 to filter for invoices dated in the first quarter of 2002 (January 1, 2002 to March 31, 2002).

   Hint: Join to the Invoices table to code a search condition based on InvoiceDate.
6. Write a SELECT statement that answers the following question: What is the total amount invoiced for each AccountNo? Use the WITH ROLLUP operator to include a row that gives the grand total.

   Hint: Use the InvoiceLineItemAmount column of the InvoiceLineItems table.

7. Write a SELECT statement that returns four columns: VendorName, AccountDescription, LineItemCount, and LineItemSum. LineItemCount is the row count, and LineItemSum is the sum of the InvoiceLineItemAmount column. For each vendor and account, return the number and sum of line items, sorted first by vendor, then by account description.

   Hint: Use a four-way join.

8. Write a SELECT statement that answers this question: Which vendors are being paid from more than one account? Return two columns: the vendor name and the total number of accounts that apply to that vendor's invoices.

   Hint: Use the DISTINCT keyword to count InvoiceLineItems.AccountNo.

Chapter 6
How to code subqueries

Objectives

- Given a specification for a result set that requires any of the language elements presented in this chapter, write a subquery and a query that generate the result set.
- List the four ways to introduce a subquery in a SELECT statement.
- Explain where a subquery can be introduced if it returns: (1) a single value; (2) a single column; (3) multiple columns.
- Describe the relative strengths of subqueries compared to joins.
- Explain the difference between a correlated subquery and a noncorrelated subquery.
- Given a SELECT statement that uses a subquery, explain how the result set of the subquery will affect the final result set.

Summary

- A *subquery* is a SELECT statement that's coded within another SQL statement.
- A subquery can return a single value, a single column, or multiple columns.
- If the subquery returns a single value, it can be *introduced* anywhere an expression is allowed. If it returns a single column, it can be introduced in place of a list of values. And if it includes one or more columns, it can be introduced in place of a table.
- The most common use for a subquery is as a *subquery search condition* in the WHERE or HAVING clause of a SELECT statement. However, subqueries are also sometimes introduced in FROM and SELECT clauses.
- Most subqueries can be restated as joins and most joins can be restated as subqueries. While joins tend to run more efficiently, subqueries are often easier to understand.
- If the ALL, ANY, or SOME keyword is coded with a subquery in a search condition, the subquery can return a list of values. Then, ALL specifies that a comparison condition must be true for all of those values; ANY or SOME specifies that the condition must be true for at least one of the values.
- A *noncorrelated subquery* is executed only once for the entire query.
- A *correlated subquery* is executed once for each row that's processed by the outer query and refers to a column in that row. Because the value of the column changes depending on the row that's being processed, the subquery returns a different result each time it's executed.
- You can use the EXISTS operator to see whether any rows will be returned by a subquery. It's usually used with correlated subqueries.

- When a subquery is coded in the FROM clause, the result set is called a *derived table* and you can use it within the outer query just as you would any table.

Terms

subquery
introduce (a subquery)
subquery search condition
subquery predicate
nested subquery
correlated subquery
noncorrelated subquery
derived table
pseudocode

Self-study questions

1. What's a subquery?
2. What are four ways a subquery can be introduced in a SELECT statement?
3. Where can a subquery be introduced if it returns a single value? a single column? multiple columns?
4. You can usually get the same result set using either a join or a subquery. What are the advantages of using a subquery? What are the advantages of using a join?
5. Consider this SQL code:

   ```
   WHERE InvoiceTotal > ALL
       (Select InvoiceTotal
       FROM Invoices
       WHERE VendorID = 34)
   ```

   When is this search condition true?
6. Consider this SQL code:

   ```
   WHERE InvoiceTotal > ANY
       (Select InvoiceTotal
       FROM Invoices
       WHERE VendorID = 34)
   ```

   When is this search condition true?
7. What's the difference between a correlated subquery and a noncorrelated subquery?
8. What does the EXISTS operator do when it's used with a subquery search condition?
9. What's a derived table?

Exercises

1. Write a SELECT statement that returns the same result set as this SELECT statement. Substitute a subquery in a WHERE clause for the inner join.

   ```
   SELECT DISTINCT VendorName
   FROM Vendors JOIN Invoices
       ON Vendors.VendorID = Invoices.VendorID
   ORDER BY VendorName
   ```

2. Write a SELECT statement that answers this question: Which invoices have a PaymentTotal that's greater than the average PaymentTotal for all paid invoices? Return the InvoiceNumber and InvoiceTotal for each invoice.
3. Write a SELECT statement that answers this question: Which invoices have a PaymentTotal that's greater than the median PaymentTotal for all paid invoices? (The median marks the midpoint in a set of values; an equal number of values lie above and below it.) Return the InvoiceNumber and InvoiceTotal for each invoice.

   Hint: Begin with the solution to exercise 2, then use the ALL keyword in the WHERE clause and code "TOP 50 PERCENT PaymentTotal" in the subquery.
4. Write a SELECT statement that returns two columns from the GLAccounts table: AccountNo and AccountDescription. The result set should have one row for each account number that has never been used. Use a correlated subquery introduced with the NOT EXISTS operator. Sort the final result set by AccountNo.
5. Write a SELECT statement that returns four columns: VendorName, InvoiceID, InvoiceSequence, and InvoiceLineItemAmount for each invoice that has more than one line item in the InvoiceLineItems table.

   Hint: Use a subquery that tests for InvoiceSequence > 1.
6. Write a SELECT statement that returns a single value that represents the sum of the largest unpaid invoices submitted by each vendor. Use a derived table that returns MAX(InvoiceTotal) grouped by VendorID, filtering for invoices with a balance due.
7. Write a SELECT statement that returns the name, city, and state of each vendor that's located in a unique city and state. In other words, don't include vendors that have a city and state in common with another vendor.
8. Write a SELECT statement that returns four columns: VendorName, InvoiceNumber, InvoiceDate, and InvoiceTotal. Return one row per vendor, representing the vendor's invoice with the earliest date.

Chapter 7
How to insert, update, and delete data

Objectives

- Given a specification for an action query, write the INSERT, UPDATE, or DELETE statement.
- Code the INTO clause of the SELECT statement to create a copy of a table.
- List and describe the three types of action queries.
- Explain how to handle null values and default values when coding INSERT and UPDATE statements.
- Explain how the FROM clause is used in an UPDATE or DELETE statement.
- Given an action query that uses any of the language elements presented in this chapter, describe what action will be taken and explain what each clause of the query does.

Summary

- You can use the SELECT INTO statement to create simple test tables that don't include keys, default values, and so on. Then, SQL Server creates a table with the name you specify and saves the result set to it.
- The INSERT statement adds a new row to a table using the column values you code in the VALUES clause.
- The values you code for the new row depend on whether or not you specify a column list in the statement. If not, you need to code values for all the columns in the table, except for identity columns, in the order that the columns appear in the table.
- If you specify a column list, you only need to provide values for the columns you've listed. The list can't include identity columns, and you can omit columns that have default values or that accept nulls.
- You can use a subquery in place of the VALUES clause if you want to add rows from another table.
- The UPDATE statement modifies selected rows in a table using the specifications you provide in the SET clause.
- You can use subqueries in the SET and WHERE clauses of the UPDATE statement and in the WHERE clause of the DELETE statement to control various aspects of the operation.
- The FROM clause of the UPDATE and DELETE statements is a SQL Server extension. In it, you can code a subquery or join to base the operation on the data in a table other than the one given in the UPDATE or DELETE clause.
- The DEFAULT and NULL keywords allow you to assign a default or null value to a new or updated column.

- The DELETE statement deletes one or more rows from a table. As with the UPDATE statement, you can use subqueries and joins to help identify the rows to be deleted.

Terms

There are no terms for this chapter.

Self-study questions

1. How do you save a result set to a new table? How would you use this technique to create a complete copy of a table?
2. What are the three types of action queries? What effect does each one have on a base table?
3. When you're adding a new row, how do you specify the column values for the row if the statement includes a column list? if it doesn't include a column list?
4. How do you handle the value of an identity column for a new row?
5. How do you assign null values and default values when you're updating a row?
6. How do you add rows from another table to the base table you're working with?
7. What's the function of the SET clause in an UPDATE statement?
8. How can you use the FROM clause in an UPDATE or DELETE statement?

Exercises

1. Write SELECT INTO statements to create two test tables named VendorCopy and InvoiceCopy that are complete copies of the Vendors and Invoices tables. If VendorCopy and InvoiceCopy already exist, first code two DROP TABLE statements to delete them.
2. Write an INSERT statement that adds a row to the InvoiceCopy table with the following values:

VendorID:	32	InvoiceTotal:	$434.58	TermsID:	2
InvoiceNumber:	AX-014-027	PaymentTotal:	$0.00	InvoiceDueDate:	11/8/02
InvoiceDate:	10/21/02	CreditTotal:	$0.00	PaymentDate:	null

3. Write an INSERT statement that adds a row to the VendorCopy table for each non-California vendor in the Vendors table. (This will result in duplicate vendors in the VendorCopy table.)
4. Write an UPDATE statement that modifies the VendorCopy table. Change the default account number to 403 for each vendor that has a default account number of 400.
5. Write an UPDATE statement that modifies the InvoiceCopy table. Change the PaymentDate to today's date and the PaymentTotal to the balance due for each invoice with a balance due. Set today's date with a literal date string, or use the GETDATE() function.

6. Write an UPDATE statement that modifies the InvoiceCopy table. Change TermsID to 2 for each invoice that's from a vendor with a DefaultTermsID of 2. Use a subquery.
7. Solve exercise 6 using a join rather than a subquery.
8. Write a DELETE statement that deletes all vendors in the state of Minnesota from the VendorCopy table.
9. Write a DELETE statement for the VendorCopy table. Delete the vendors that are located in states from which no vendor has ever sent an invoice.

   Hint: Use a subquery coded with "SELECT DISTINCT VendorState" introduced with the NOT IN operator.

Chapter 8
How to work with data types and functions

Objectives

- Given a specification for a query that requires any of the language elements presented in this chapter, write the query statement.
- Given a column of a particular data type, code a CAST and a CONVERT function to explicitly convert to any of the other applicable data types.
- Name the four data type categories and explain what kind of information they store.
- Given any SQL Server data type, describe what kind of information it stores.
- Explain the difference between standard character data and Unicode character data.
- Explain the difference between implicit and explicit data type conversion.
- Given a query that uses any of the language elements presented in this chapter, identify each column's data type and explain what each function does.

Summary

- There are four categories of data types in SQL. *String data types* are designed to store character, or text, data. *Numeric data types* are designed to store numbers. *Temporal (date/time) data types* are designed to store dates, times, or both. And the remaining data types are used to store a variety of data items, like binary data or system pointers.
- *Integer data types* are used to store whole numbers; *decimal data types* are used to store values with fractional portions; and *real data types* are used to store *floating-point numbers* that have a limited number of *significant digits*.
- The *precision* of a decimal value indicates the total number of digits that can be stored, while the *scale* indicates how many digits can be stored to the right of the decimal point.
- The string data types can be used to store standard characters that use a single byte of storage or *Unicode characters* that use two bytes of storage. Because Unicode data requires more storage, it's typically used only in multi-language environments, to support a wider range of characters.
- The char and nchar data types are used to store *fixed-length strings*, which always use the same amount of storage. In contrast, the varchar and nvarchar types are used for *variable-length strings*, where the amount of storage varies depending on the size of the string.
- Date/time values are stored with an integer portion that represents the number of days since a base date and a fractional portion that represents the elapsed time since midnight.

- To specify a date/time value, you can code a date/time literal in quotes that conforms to a date/time format that's recognized by your system.
- If you don't specify a time when storing a date value, the time defaults to 12:00 a.m. If you don't specify a date when storing a time value, the date defaults to January 1, 1900.
- To compare data values or use them in calculations, the data must have matching data types. SQL Server often converts data automatically according to its order of precedence, converting the data with the lower data type to a higher data type. This is called *implicit conversion*.
- If SQL Server can't handle the conversion automatically or if you want to convert data with a higher data type to a lower data type, you can use the CAST or CONVERT function to do an *explicit conversion*. SQL Server also provides other functions that perform special types of conversions.
- Many SQL Server functions let you handle various types of data. For example, you can use string functions to count the characters in a string or to parse long strings into shorter elements. You can use numeric functions to round numbers or find a square root. And you can use date/time functions to find the day, month, or year portion of a value or to compare two dates to get the elapsed time.

Terms

data type
string data type
numeric data type
temporal data type
date/time data type
date data type
BLOB (binary large object)
integer data type
decimal data type
scale
precision
real data type
fixed-point number
floating-point number
significant digits
single-precision number
double-precision number
scientific notation
exact numeric data types
approximate numeric data types
Unicode character
Unicode specification
national character
fixed-length string
variable-length string
bit
ASCII (American Standard Code for Information Interchange)
implicit conversion
explicit conversion
cast
Universal Time Cordinate (UTC)
Greenwich Mean Time

Self-study questions

1. The SQL Server data types can be divided into four categories. What are they? What type of data does each one store?
2. What type of information is stored by each of the following data types:
   a. varchar
   b. int
   c. decimal
   d. real
   e. datetime
3. What's the difference between fixed-point and floating-point numbers?
4. When you're working with decimal data, what does *precision* mean? *scale*?
5. What's the difference between standard character data and Unicode character data?
6. In general terms, how are date/time values stored? Does SQL Server provide a data type for storing just the date without the time?
7. What's the difference between implicit and explicit data conversion? How do you do an explicit conversion?
8. What's the difference between the CAST and CONVERT functions?
9. What are some of the operations you can perform with string functions? What are two common problems that occur with string data?
10. What two numeric functions are you most likely to use? How do they work? What are two common problems that occur with numeric data?
11. What are some of the operations you can perform with date/time functions? What's the major problem you run into when you're searching date/time values for a specific date or time?
12. How does the CASE function work?
13. When would you use the COALESCE or ISNULL function?
14. What's a common use for the GROUPING function?

Exercises

1. Write a SELECT statement that returns four columns based on the InvoiceTotal column of the Invoices table:
   - Use the CAST function to return the first column as data type decimal with 2 digits to the right of the decimal point.
   - Use CAST to return the second column as a varchar.
   - Use the CONVERT function to return the third column as the same data type as the first column.
   - Use CONVERT to return the fourth column as a varchar, using style 1.
2. Write a SELECT statement that returns four columns based on the InvoiceDate column of the Invoices table:
   - Use the CAST function to return the first column as data type varchar.
   - Use the CONVERT function to return the second and third columns as a varchar, using style 1 and style 10, respectively.
   - Use the CAST function to return the fourth column as data type real.
3. Write a SELECT statement that returns two columns based on the Vendors table. The first column, Contact, is the vendor contact name in this format: first name followed by last initial (for example, "John S.") The second column, Phone, is the VendorPhone column without the area code. Only return rows for those vendors in the 559 area code. Sort the result set by first name, then last name.
4. Write a SELECT statement that returns the InvoiceNumber and balance due for every invoice with a non-zero balance and an InvoiceDueDate that's less than 30 days from today.
5. Modify the search expression for InvoiceDueDate from the solution for exercise 4. Rather than 30 days from today, return invoices due before the last day of the current month.
6. Write a summary query WITH CUBE that returns LineItemSum (which is the sum of InvoiceLineItemAmount) grouped by Account (an alias for AccountDescription) and State (an alias for VendorState). Use the CASE and GROUPING function to substitute the literal value "*ALL*" for the summary rows with null values.
7. *(If you have access to the Examples database)* Modify the third SELECT statement shown in figure 8-11 of the text to return a middle name, if present. Add a third column, Middle, which is null if no middle name is present.

Chapter 9
How to design a database

Objectives

- Given a specification for a database modeled on a real-world system, design the database. Identify tables, columns, keys, relationships, and indexes for the new database.
- Given a diagram for an unnormalized database, normalize the structure to the third normal form.
- Given a diagram for a database normalized to the fifth or sixth normal form, identify which tables and columns in the structure should be eliminated to denormalize the structure for optimal performance.
- In general terms, describe the criteria for indexing a column.
- Explain how referential integrity prevents deletion, insertion, and update anomalies.
- Explain how normalization reduces data redundancy.
- Identify the normal form at which a database is considered strictly normalized and the normal form at which most designers consider the structure normalized.
- Briefly describe the criteria for the first, second, and third normal forms.
- Explain why denormalization can sometimes improve system performance.

Summary

- When designing a database, the DBA or programmer usually designs its *data structure* based on the real-world system for which it will be used.
- A table in a relational database typically represents an object, or *entity*, in the real world. Each column of a table is used to store an *attribute* associated with the entity, and each row represents one *instance* of the entity.
- In general, to design a database, you do the following: identify the data elements; subdivide each element into its smallest useful components; group the elements into tables and assign the columns; identify the primary and foreign keys; review whether the data structure is normalized; and identify the columns that should be indexed.
- The *referential integrity* of a database depends on the relationships between its tables being maintained correctly when rows are deleted, updated, or inserted. You can maintain referential integrity using either *foreign key constraints* or *triggers*.
- An *index* provides for locating one or more rows directly, without scanning all the data in the table. A *clustered index* defines the sequence in which table rows are stored, so there can be only one per table. You can also define *unclustered indexes*.
- To *normalize* a data structure, you use a formal process to separate the data into related tables. This reduces *data redundancy*, which can cause storage and maintenance problems.

- The normalization process consists of *normal forms* that are applied in sequence. Although there are seven normal forms, a data structure is typically considered normalized if the first three are applied.
- Because a completely normalized data structure can be inefficient and difficult to code and debug, most structures are *denormalized* whenever the increased efficiency will outweigh the potential for redundancy errors and storage problems.

Terms

data structure
entity
attribute
instance
entity-relationship (ER) modeling
CASE (computer-aided software engineering)
linking table
connecting table
associate table
referential integrity
declarative referential integrity (DRI)
foreign key constraints
triggers
orphaned row
update anomaly
insertion anomaly
deletion anomaly
normalization
data redundancy
unnormalized data structure
normalized data structure
normal forms
index
table scan
linked list
page
node
page header
index row
clustered index
nonclustered index
composite index
covering index
Boyce-Codd normal form
transitive dependency
multivalued dependency
domain-key normal form
derived data
denormalized data structure
denormalization

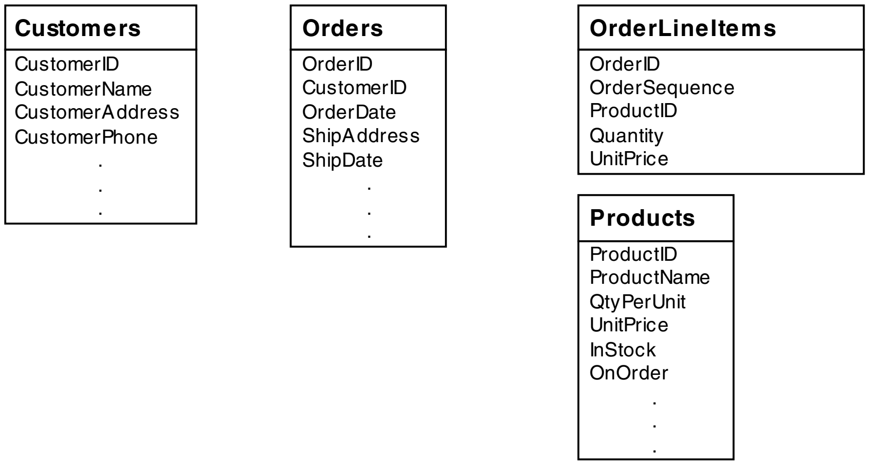
Self-study questions

1. What are six basic steps you can follow to design a data structure?
2. What should the data structure be based on? In light of that, how is an *entity* represented in the resulting database? How is an *attribute* of the entity represented? How is an *instance* of the entity represented?
3. What are some resources you can use to identify the data elements of the data structure?
4. How can you identify the primary keys in a database?
5. What does *referential integrity* mean? What happens when referential integrity isn't enforced? What are two methods you can use to maintain referential integrity in a database?
6. What guidelines can help you decide whether to create an index for a table?
7. What's the difference between a clustered and nonclustered index?
8. What is *normalization* of a data structure? How does it reduce data redundancy?

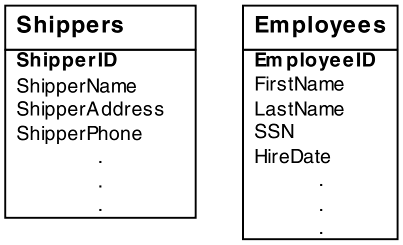
9. How many normal forms are there? A database is considered strictly normalized at which normal form? In practical terms, most designers consider a database normalized at which normal form?
10. What are the main criteria for the first normal form? the second normal form? the third normal form?
11. How can denormalization improve system performance?

Exercises

1. Design a database diagram for a product orders database with four tables. Indicate the relationships between tables and identify the primary key and foreign keys in each table. Explain your design decisions.

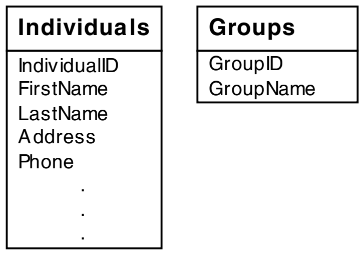


2. Add the two tables below into the design for exercise 1. Create additional tables and columns, if necessary. Explain your design decisions.



3. Modify your design for exercise 2 to identify the columns that should be indexed, and explain your decision.

4. Design a database diagram that allows individuals to be assigned membership in one or more groups. Each group can have any number of individuals and each individual can belong to any number of groups. Create additional tables and columns, if necessary. Explain your design decisions.



5. Modify your design for exercise 4 to keep track of the *role* served by each individual in each group. Each individual can only serve one role in each group. Each group has a unique set of roles that members can fulfill. Create additional tables and columns, if necessary. Explain your design decisions.

Chapter 10
How to create and maintain databases and tables

Objectives

- Given a complete database design, write the SQL DDL statements to create the database, including all tables, relationships, constraints, and indexes.
- Given a name for a SQL Server database object, determine whether or not the identifier must be delimited.
- Explain the behavior of a column with the IDENTITY attribute when a new row is inserted into the table.
- Briefly describe how each of the five types of constraints restrict the values that can be stored in a table.
- Describe the main difference between a column-level constraint and a table-level constraint.
- Explain how the CASCADE and NO ACTION options differ in enforcing referential integrity on deletes and updates.
- Given a series of SQL DDL statements that use any of the language elements presented in this chapter, explain what each statement does.

Summary

- As an application programmer, you'll use the *data definition language (DDL)* statements to create, modify, and delete the database objects that make up your test databases.
- Each database object is named using an *identifier*. There are standard rules for creating identifiers. However, if you have to work with an identifier that doesn't follow the rules, you can code a delimited identifier for the object in your SQL statements.
- You can use the CREATE DATABASE statement to create a new, empty database. You can also use it to *attach* a database that was created on another server.
- The CREATE TABLE statement defines the columns in a table. That includes identifying the primary key and identity columns and specifying which columns allow nulls or have a default value.
- SQL Server automatically creates a clustered index for a table based on its primary key and a nonclustered index for each unique key other than the primary key. To create additional nonclustered indexes, you use the CREATE INDEX statement.
- *Constraints* are rules that are defined at the table or column level to enforce the integrity of the data in the table.

- A *check constraint* limits the values that can be stored in a column. A *foreign key constraint* defines the relationship between a primary and foreign key table and enforces referential integrity.
- A foreign key constraint can specify that updates or deletions to one table be *cascaded* to the related table. Or it can specify that updates and deletions aren't allowed when there are related rows, so an error is raised.
- The DROP and ALTER statements are used to delete and modify existing database objects.

Terms

data definition language (DDL)
database objects
identifier
log file
attach a database
constraint
column-level constraint
table-level constraint
check constraint
foreign key constraint
reference constraint
cascading delete
cascading update
script
batch

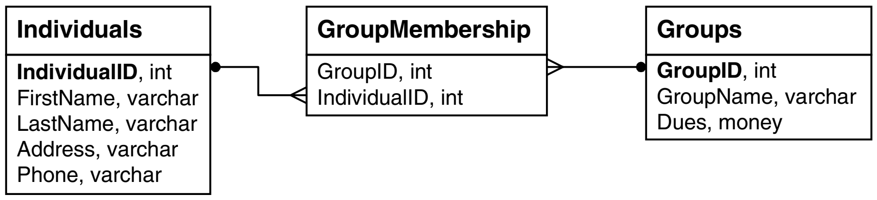
Self-study questions

1. What is the function of the DDL statements in SQL?
2. When does an object identifier need to be delimited? Which of the following identifiers need to be delimited? Create a delimited identifier where needed.
   a. Invoices
   b. InvoicesTable
   c. Invoices Table
   d. Invoices_Table
   e. $Invoices
   f. [Invoices Table]
3. What two files does a CREATE DATABASE statement create?
4. If you copy a database from another server, what do you have to do before you can access it? What statement do you use?
5. What statement do you use to define the columns in a table?
6. If a column in a table is defined with the IDENTITY attribute, what happens when a new row is added to the table?
7. What are the five types of constraints you can use in defining a table? How does each one affect the values that can be stored in the table?
8. What's the difference between a column-level constraint and a table-level constraint?
9. What happens if a row that's being added or updated doesn't meet all of the constraints for the table?

10. How do the CASCADE and NO ACTION options differ in enforcing referential integrity on deletes and updates?
11. If you've created an index, table, or database to use for testing, how do you delete it once you're done with it?
12. How do you make changes to the design of a table that you're using only for testing?

Exercises

1. Create a new database named Membership.
2. Write the CREATE TABLE statements needed to implement the following design in the Membership database. Include reference constraints. Define IndividualID and GroupID with the IDENTITY keyword. Decide which columns should allow null values, if any, and explain your decision. Define the Dues column with a default of zero and a check constraint to allow only positive values.



3. Write the CREATE INDEX statements to create a clustered index on the GroupID column and a nonclustered index on the IndividualID column of the GroupMembership table.
4. Write an ALTER TABLE statement that adds a new column, DuesPaid, to the Individuals table. Use the bit data type, disallow null values, and assign a default Boolean value of False.
5. Write an ALTER TABLE statement that adds two new check constraints to the Invoices table of the AP database. The first should allow (1) PaymentDate to be null only if PaymentTotal is zero and (2) PaymentDate to be not null only if PaymentTotal is greater than zero. The second constraint should prevent the sum of PaymentTotal and CreditTotal from being greater than InvoiceTotal.
6. Delete the GroupMembership table from the Membership database. Then write a CREATE TABLE statement that recreates the table, this time with a unique constraint that prevents an individual from being a member in the same group twice.

Chapter 11
How to use the Enterprise Manager

Objectives

- Given a SQL query, use the Enterprise Manager to create the query, using only the diagram and grid panes, if possible.
- Given a complete database design, use the Enterprise Manager to create the database, including all tables, relationships, constraints, and indexes.
- Use the Enterprise Manager to create a backup copy of a database.
- Use the Enterprise Manager to restore a database from a backup copy.
- Given a particular table in a database, view all of the objects that are dependent on the table.
- Create, save, and print a database diagram of a database using the Enterprise Manager.
- Compare the Enterprise Manager to the Query Analyzer, and describe what tasks are better performed using each tool.
- Name the four panes of the Query Designer window, and describe the function of each.
- Explain the difference between a complete backup and a differential backup.
- Explain what table dependencies are and why you have to be aware of them.

Summary

- The Enterprise Manager is a graphical tool that you can use to generate and execute SQL code.
- You can use the *Query Designer* to create or edit a query. In its *diagram pane,* you select the tables and columns you want to retrieve and then define any further column specifications in the *grid pane*. The *SQL pane* shows the resulting SQL code, and the *results pane* shows the result set. You can also add, modify, or delete rows in the results pane.
- You can also use the Query Designer to create and execute action queries. In that case, a dialog box displays how many rows were affected by the query.
- Instead of coding DDL statements, you can use the Enterprise Manager to more easily create and work with databases, tables, indexes, and keys.
- You can also easily back up and restore a database. The backup can be a complete backup or a *differential backup* that contains just the information that's changed since the last backup.
- The Enterprise Manager also allows you to look at the relationships within a database. You can view the table *dependencies* to see which database objects a

selected table depends on and which objects depend on it. And you can view the *database diagram* to see how the tables in the database are related.

Terms

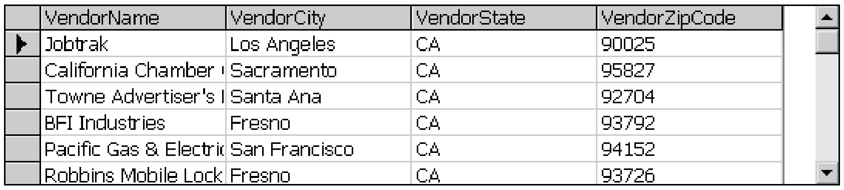
Enterprise Manager
node
Query Designer
diagram pane
grid pane
SQL pane
results pane
differential backup
replication
dependencies
first-level dependency
database diagram
Database Designer

Self-study questions

1. If you compare the Enterprise Manager to the Query Analyzer, what tasks are easier to do using each tool?
2. What are the four panes of the Query Designer window? How do you use each one?
3. How do you create an action query using the Query Designer? In other words, how does it differ from creating a SELECT query?
4. What's the difference between a complete backup and a differential backup?
5. In general terms, what information do you have to provide to back up a database using the Enterprise Manager? to restore a database?
6. Which of the following can you do using the Enterprise Manager?
   a. attach a database to your server
   b. define a column as an identity column
   c. delete a column from a table
   d. create a foreign key constraint
   e. set up a one-to-many relationship between two tables
   f. create a nonclustered index
7. What are table dependencies? Why do you have to be aware of them?
8. What's a database diagram? How does it show the relationships between tables?

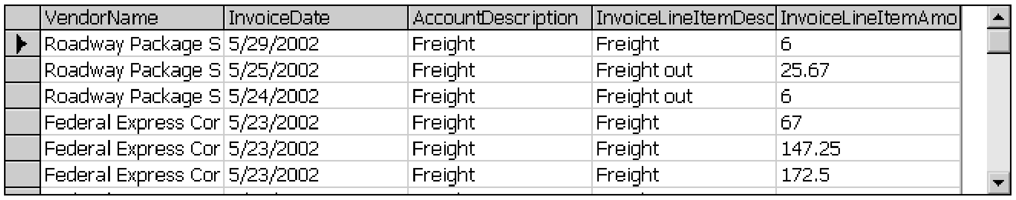
Exercises

1. Using the diagram and grid panes of the Query Designer, query the Vendors table for vendors in California. The results pane should look like this:



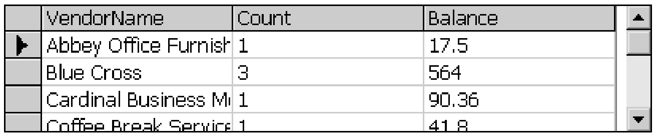
VendorName	VendorCity	VendorState	VendorZipCode
Jobtrak	Los Angeles	CA	90025
California Chamber	Sacramento	CA	95827
Towne Advertiser's	Santa Ana	CA	92704
BFI Industries	Fresno	CA	93792
Pacific Gas & Electri	San Francisco	CA	94152
Robbins Mobile Lock	Fresno	CA	93726

2. Using the diagram and grid panes of the Query Designer, create a query that joins the Vendors, Invoices, InvoiceLineItems, and GLAccounts tables. Filter the result set for rows with AccountNo 553, and sort the result set with the most recent InvoiceDate first.



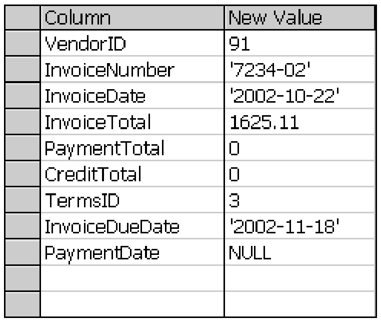
VendorName	InvoiceDate	AccountDescription	InvoiceLineItemDesc	InvoiceLineItemAmo
Roadway Package S	5/29/2002	Freight	Freight	6
Roadway Package S	5/25/2002	Freight	Freight out	25.67
Roadway Package S	5/24/2002	Freight	Freight out	6
Federal Express Cor	5/23/2002	Freight	Freight	67
Federal Express Cor	5/23/2002	Freight	Freight	147.25
Federal Express Cor	5/23/2002	Freight	Freight	172.5

3. Using the diagram and grid panes of the Query Designer, create a summary query that joins Vendors and Invoices. For each vendor, return the number of invoices with an outstanding balance and the sum of the outstanding balances. Determine whether the search condition appears in the WHERE clause or the HAVING clause, then change the grid pane to switch where it appears.



VendorName	Count	Balance
Abbey Office Furnish	1	17.5
Blue Cross	3	564
Cardinal Business M	1	90.36
Coffee Break Servic	1	41.8

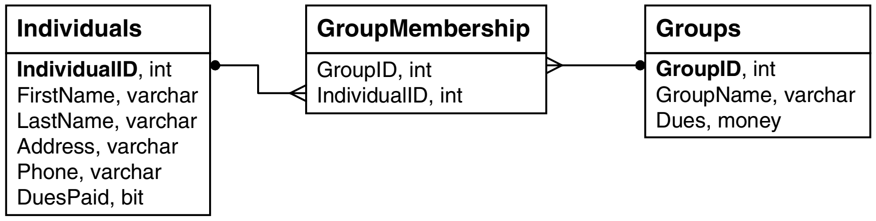
4. Use the Query Designer to create a SELECT INTO query that makes a copy of the Invoices table called InvoiceCopy2. Then, use the Query Designer to insert a new row into the InvoiceCopy2 table using the following values:



Column	New Value
VendorID	91
InvoiceNumber	'7234-02'
InvoiceDate	'2002-10-22'
InvoiceTotal	1625.11
PaymentTotal	0
CreditTotal	0
TermsID	3
InvoiceDueDate	'2002-11-18'
PaymentDate	NULL

5. a. Use the Enterprise Manager to create a new database called Membership using the default settings. (If the database already exists, use the Enterprise Manager to drop it, then recreate it.)

   b. Use the Enterprise Manager to create the following tables and relationships in the Membership database. Define IndividualID and GroupID as IDENTITY columns. Allow Address and Phone to accept null values; none of the other columns should allow null values. Define the Dues column with a default of zero and a check constraint to allow only positive values. Define the DuesPaid column with a default Boolean value of False.



   c. Use the Enterprise Manager to index the GroupMembership table. Create a clustered index on the GroupID column, a nonclustered index on the IndividualID column, and a unique index and constraint on both columns.

   d. Use the Enterprise Manager Database Designer window to create and print a database diagram for the completed Membership database.

Chapter 12
How to work with views

Objectives

- Given a SELECT statement, create a new view based on the statement.
- Given a SELECT statement, create a new view based on the statement using the Enterprise Manager.
- Change the design of an existing view using the Enterprise Manager.
- Given a SELECT statement, determine whether it can be used as the basis for a view. Explain why.
- Given a SELECT statement, determine whether it can be used as the basis for an updatable view. Explain why.
- Explain how a view can be used to restrict access to sensitive data.
- Explain how a view can be used to simplify the database structure seen by a user.
- Explain the effect of the WITH SCHEMABINDING and the WITH ENCRYPTION clauses on a view.
- Explain the effect of the WITH CHECK OPTION clause on an updatable view.
- Explain what information you can expect to find in the system catalog.

Summary

- A *view*, also referred to as a *viewed table*, consists of a SELECT statement that's stored as an object in the database. The tables that the SELECT statement refers to are called the *base tables* for the view.
- You can refer to a view anywhere you would normally use a table in any of the data manipulation statements.
- Views provide a variety of benefits, including design independence, data security, flexibility, simplified queries, and updatability.
- The SELECT statement you code within the definition of a view can refer to another view. However, it can't include the INTO keyword, and it can include an ORDER BY clause only if it also includes the TOP keyword.
- To create a view, you use the CREATE VIEW statement. On this statement, you can provide the names for the columns in the view. You can also provide column names using the AS clause in the SELECT statement. If you don't name the columns in a view, the column names from the base table are used.
- The CREATE VIEW statement provides three optional clauses. The WITH ENCRYPTION clause prevents other users from examining the SELECT statement for the view. The WITH SCHEMABINDING clause prevents the underlying base tables from being deleted or modified in a way that affects the view. And the WITH

CHECK OPTION clause prevents a row in a view from being updated if it would cause the row to be excluded from the view.

- An *updatable view* is one that can be used in an INSERT, UPDATE, or DELETE statement. If a view isn't updatable, it's called a *read-only view*.
- For a view to be updatable, SQL Server must be able to unambiguously determine which base tables and which columns are affected.
- To delete a view, you use the DROP VIEW statement. When you delete a view, any procedures or triggers that are dependent on it and any permissions that are assigned to it are also deleted.
- To modify the definition of a view, you can delete the view and then create it again, or you can use the ALTER VIEW statement to specify the new definition.
- You can use the *information schema views* provided by SQL Server to examine the *system catalog*, which lists all of the system objects that define a database, including tables, views, columns, keys, and constraints.
- To design a new view using the Enterprise Manager, you use the New View window. To modify an existing view, you use the Design View window. You can also display and modify a view and set its permissions using the View Properties window.
- You can use the Properties dialog box for a view to add a WITH CHECK OPTION, WITH SCHEMA BINDING, or WITH ENCRYPTION clause to a view.

Terms

view	nested view	read-only view
viewed table	database schema	information schema view
base table	updatable view	system catalog

Self-study questions

1. If the SELECT statement for a view includes an ORDER BY clause, what other clause must it include?
2. Why can't the SELECT statement for a view include an INTO clause?
3. When must you provide a name for a column in a view? What two techniques can you use to specify that name?
4. What is an updatable view?
5. When you're coding the select list for an updatable view, what two clauses are you not allowed to use? What other coding restrictions are placed on the select list?
6. In the SELECT statement for an updatable view, there are two additional clauses you're not allowed to use. What are they? What operator is prohibited?
7. How might you code a view that's used to restrict access to sensitive data? a view that simplifies the database structure seen by the user?
8. Why would you code the WITH SCHEMABINDING clause on a CREATE VIEW statement? the WITH ENCRYPTION clause?

9. Under what situation will an error occur if you code the WITH CHECK OPTION clause on the CREATE VIEW statement?
10. What is the system catalog? What type of information does it contain?

Exercises

1. Write a CREATE VIEW statement that defines a view named InvoiceBasic that returns three columns: VendorName, InvoiceNumber, and InvoiceTotal. Then, write a SELECT statement that returns all of the columns in the view, sorted by VendorName, where the first letter of the vendor name is N, O, or P.
2. Create a view named Top10PaidInvoices that returns three columns for each vendor: VendorName, LastInvoice (the most recent invoice date), and SumOfInvoices (the sum of the InvoiceTotal column). Return only the 10 vendors with the largest SumOfInvoices and include only paid invoices.
3. Create an updatable view named VendorAddress that returns the VendorID, both address columns, and the city, state, and zip code columns for each vendor. Then, write a SELECT query to examine the result set where VendorID=4. Next, write an UPDATE statement that changes the address so that the suite number (Ste 260) is stored in VendorAddress2 rather than in VendorAddress1. To verify the change, rerun your SELECT query.
4. *(If you've done exercise 10-5)* Query each of the four information schema views listed in figure 12-9 of the text that return information about constraints. Which of these four views returns the code for a check constraint clause? What is the name of the column in which the code is returned?
5. Using the Enterprise Manager, create a view named AccountByVendor that returns the sum of InvoiceLineItemAmounts in the InvoiceLineItems table, grouped by VendorName and AccountDescription. Check the joins that are automatically coded in the FROM clause to be sure they're correct.
6. Using the Enterprise Manager, modify the InvoiceBasic view created in exercise 1 to sort the result set by VendorName. What clause does the system automatically code to allow the use of an ORDER BY clause in the view?

Chapter 13
How to code scripts

Objectives

- Given a Transact-SQL script written as a single batch, insert GO commands to divide the script into appropriate batches.
- Given the specification for a database problem, write a script that solves it.
- Use OSQL to execute a query against a database.
- Explain the scope of a local variable.
- Describe the difference between a scalar variable and a table variable.
- Compare and contrast the scopes of temporary tables, table variables, and derived tables.
- Describe the difference between dynamic SQL and ordinary SQL code.
- Given a Transact-SQL script, explain what each statement does.

Summary

- A *script* is a series of SQL statements that can be divided into one or more *batches*. To indicate the end of a batch, you code a GO command.
- Within a script, you can use Transact-SQL statements (also known as *T-SQL statements*) that are specifically designed to handle procedural processing.
- To store data within a script, you use the DECLARE statement to declare a *variable*. A variable that can contain a single value is called a *scalar variable*. A variable that can contain an entire result set is called a *table variable*.
- Because variables can only be used within the batch in which they're defined, they're often referred to as *local variables*.
- You can use the SET statement to assign a value to a variable, or you can use a SELECT statement to assign a value to one or more variables.
- To create a table variable, you use a DECLARE statement with the table data type. Then, you define the columns of the table.
- You can use a table variable almost anywhere you would normally code a table name. The exception is that you can't code it instead of a table name in the INTO clause of a SELECT INTO statement.
- A *temporary table* is a table that exists only during the current database session. A *local temporary table* is visible only within the current session. A *global temporary table* is visible to all sessions.
- You can use the IF…ELSE statement to test for a condition within a script. Then, the processing that's done depends on whether that condition is true or false.

- You can use the OBJECT_ID function within an IF statement to check for the existence of a table, view, stored procedure, user-defined function, or trigger. You can use the DB_ID function within an IF statement to check for the existence of a database.
- You can use the WHILE statement to perform one or more statements repeatedly as long as a condition is true. To execute two or more statements within a WHILE *loop*, you enclose them with BEGIN and END keywords.
- To exit from a WHILE loop immediately, you use the BREAK statement. To return to the beginning of a WHILE loop immediately, you use the CONTINUE statement.
- You can use *system functions* to return information about SQL Server values, objects, and settings. System functions can be used anywhere an expression is allowed.
- You can control the way queries and scripts execute within the current session by using the SET statement to change the configuration settings.
- You can use the EXEC statement to execute a SQL statement that's defined within the script and that can change each time the script is executed. This is referred to as *dynamic SQL*.
- You can use the *OSQL* utility to enter and execute scripts from a command line.

Terms

script
batch
connection
session
T-SQL statement
variable
scalar variable
local variable
table variable
temporary table
local temporary table
global temporary table
scope
nested IF...ELSE statements
WHILE loop
system function
global variable
dynamic SQL
OSQL
ISQL

Self-study questions

1. What is the purpose of a GO command within a script? When do you need to code one?
2. What is a variable? How do you create one?
3. What is the scope of a local variable?
4. What is the difference between a scalar variable and a table variable? Where can you use each?
5. What is the scope of a temporary table? a table variable? a derived table?
6. How does a local temporary table differ from a global temporary table?
7. What is the purpose of the IF…ELSE statement? How can you check for the existence of a database object within this statement?

8. What is the purpose of the WHILE statement?
9. What type of information do the system functions return? Where can you use these functions?
10. What are some of the configuration settings that you might change using the SET statement?
11. What is dynamic SQL? When would you use it instead of ordinary SQL?
12. What is the purpose of the OSQL utility? When might you want to use this utility?

Exercises

1. Write a script that declares and sets a variable that's equal to the total outstanding balance due. If that balance due is greater than $10,000.00, the script should return a result set consisting of VendorName, InvoiceNumber, InvoiceDueDate, and Balance for each invoice with a balance due, sorted with the oldest due date first. If the total outstanding balance due is less than $10,000.00, return the message "Balance due is less than $10,000.00."
2. The following script uses a derived table to return the date and invoice total of the earliest invoice issued by each vendor. Write a script that generates the same result set but uses a temporary table in place of the derived table. Make sure your script tests for the existence of any objects it creates.

```
USE AP

SELECT VendorName, FirstInvoiceDate, InvoiceTotal
FROM Invoices JOIN
  (SELECT VendorID, MIN(InvoiceDate) AS FirstInvoiceDate
   FROM Invoices
   GROUP BY VendorID) AS FirstInvoice
  ON (Invoices.VendorID = FirstInvoice.VendorID AND
      Invoices.InvoiceDate = FirstInvoice.FirstInvoiceDate)
JOIN Vendors
  ON Invoices.VendorID = Vendors.VendorID
ORDER BY VendorName, FirstInvoiceDate
```

3. Write a script that generates the same result set as the code shown in example 2, but uses a view instead of a derived table. Also write the script that creates the view. Make sure that your script tests for the existence of the view. The view doesn't need to be redefined each time the script is executed.
4. Write a script that uses dynamic SQL to return a single column that represents the number of rows in a particular table in the current database. The script should automatically choose the user base table that appears first alphabetically. Exclude system tables, views, and the table named "dtproperties." Name the column CountOfTable, where Table is the chosen table name.

   Hint: Use one of the information schema views.

Chapter 14
How to code stored procedures, functions, and triggers

Objectives

- Given a specification for a database problem, write a stored procedure that solves it. Write a script that calls the procedure.
- Given a formula or expression, write a scalar-valued user-defined function based on it. Write a script that invokes the function.
- Given a SELECT statement with a WHERE clause, write a table-valued user-defined function that replaces it. Write a script that invokes the function.
- Given a specification for a database problem that could be caused by an action query, write a trigger that prevents it. Write a script that causes the trigger to fire.
- Explain the effect of the WITH RECOMPILE clause, the WITH ENCRYPTION clause, and the WITH SCHEMABINDING clause on a stored procedure, user-defined function, or trigger.
- Explain why you'd want to use the ALTER statement rather than dropping and recreating a procedure, function, or trigger.
- Given a stored procedure, user-defined function, or trigger, explain what each statement does.

Summary

- *Stored procedures*, *user-defined functions*, and *triggers* are executable database objects that contain SQL statements and that provide greater control and better performance than scripts.
- Stored procedures are *precompiled*, which means that the *execution plan* for the SQL code is compiled the first time the procedure is executed and is then saved in its compiled form. To *call* a stored procedure, you use the EXEC statement.
- To create a stored procedure, you use the CREATE PROC statement. You can create either a permanent stored procedure or a *temporary stored procedure* with either local or global scope.
- A stored procedure can include *input parameters* that accept values from the calling program and *output parameters* that store values that are passed back to the calling program.
- An *optional parameter* is an input parameter whose value doesn't have to be provided by the calling program. An optional parameter must have a default value.
- You can use the RETURN statement to pass a return value back to the calling program.

- You can use the RAISERROR statement to manually set an error condition. It's typically used in stored procedures that provide for *data validation*.
- To delete a stored procedure, you use the DROP PROC statement. To modify the definition of a stored procedure, you can delete the stored procedure and create it again or use the ALTER PROC statement to specify the new definition.
- SQL Server 200 includes many *system stored procedures* that you can use within the scripts and procedures you write to perform useful tasks on a database.
- A user-defined function always returns a value. A function can include input parameters, but not output parameters.
- A *scalar-valued function* returns a single value of any type and a *table-valued function* returns an entire table.
- To create a user-defined function, you use the CREATE FUNCTION statement. You specify the data type of the returned value on the RETURNS clause.
- A scalar-valued function must include a RETURN statement that specifies the value to be returned. To call, or *invoke*, a scalar-valued function, you include it in an expression. Then, the value returned by the function is substituted for the function.
- A table-valued function that's based on a single SELECT statement is called a *simple table-valued function*. You define the table to be returned by coding a SELECT statement in the RETURN statement.
- A table-valued function that's based on two or more SQL statements is called a *multi-statement table-valued function*. To return the table that's created to the calling program, you code a RETURN statement.
- To invoke a simple or multi-statement table-valued function, you refer to it anywhere you would normally code a table or view name.
- To delete a user-defined function, you use the DROP FUNCTION statement. To modify the definition of a function, you can delete the function and then create it again, or you can use the ALTER FUNCTION statement to specify the new definition.
- A trigger is a special kind of procedure that *fires* in response to an action query. Because you can't invoke a trigger directly, you can't pass values to it and a trigger can't pass back a return value.
- A trigger is associated with a single table or view and can be set to fire on INSERT, UPDATE, or DELETE statements or any combination of these statements.
- An AFTER trigger fires after the action query and is typically used to enforce referential integrity. An INSTEAD OF trigger fires instead of the action query and typically performs the required action.
- Triggers can also be used to enforce database rules for data consistency.
- To delete a trigger, you use the DROP TRIGGER statement. To modify the definition of a trigger, you can delete the trigger and then create it again, or you can use the ALTER TRIGGER statement to specify the new definition.

Terms

stored procedure
user-defined function (UDF)
trigger
sproc
call a procedure
precompiled
execution plan
recursive call
recursion
temporary stored procedure
local procedure
global procedure
parameter
input parameter
output parameter
required parameter
optional parameter
passing parameters by position
passing parameters by name
return value
data validation
system stored procedure
scalar-valued function
table-valued function
simple table-valued function
multi-statement table-valued function
invoke a function
inline table-valued function
fire a trigger
transaction

Self-study questions

1. What do you have to do to include an input parameter in a stored procedure? an output parameter? an optional parameter?
2. What statement do you use to call a stored procedure? How do you pass input parameters to a stored procedure? How do you receive output parameters?
3. How do you pass a return value back from a stored procedure? How is a return value typically used?
4. What statement can you use within a stored procedure to manually set an error condition? How is this statement typically used?
5. What happens when you run a stored procedure that was created with the WITH RECOMPILE option? Why would you not typically use this option?
6. What clause would you code when creating a stored procedure, user-defined function, or trigger to prevent users from viewing its code?
7. Under what circumstances would you want to use an ALTER statement to modify a stored procedure, user-defined function, or trigger rather than dropping and then recreating it?
8. What does the WITH SCHEMABINDING clause of the CREATE FUNCTION statement prevent you from doing?
9. Where do you code the statements that define a scalar-valued or multi-statement table-valued function? How do you return the results of one of these functions?
10. How do you create and return the result set for a simple table-valued function?
11. What are the two types of triggers that you can create? When is each one executed, and how is it typically used?

Exercises

1. Create a stored procedure in the AP database named spWhichTable that accepts a column name and returns the name of the table or tables that have a column by that name. Code a statement that calls the procedure.
2. Create a stored procedure named spBalanceRange that accepts three optional parameters. The procedure returns a result set consisting of VendorName, InvoiceNumber, and Balance for each invoice with a balance due, sorted with largest balance due first. The parameter @VendorVar is a mask that's used with a LIKE operator to filter by vendor name, as shown in figure 14-5 of the text. @BalanceMin and @BalanceMax are parameters used to specify the requested range of balances due. If called with no parameters, the procedure should return all invoices with a balance due.
3. Code three calls to the procedure in exercise 2:
   (1) passed by position with @VendorVar='Z%' and no balance range
   (2) passed by name with @VendorVar omitted and a balance range from $200 to $1000
   (3) passed by position with a balance due that's less than $200 filtering for vendors whose names begin with C or F
4. Create a stored procedure named spDateRange that accepts two parameters, @DateMin and @DateMax, with data type varchar and default value null. If called with no parameters or with null values, the procedure should return an error message describing the syntax. If called with non-null values, validate the parameters. Test that the literal strings are valid dates and test that @DateMin is earlier than @DateMax. If the parameters are valid, return a result set that includes the InvoiceNumber, InvoiceDate, InvoiceTotal, and Balance for each invoice for which the InvoiceDate is within the date range, sorted with earliest invoice first.
5. Create a scalar-valued function named fnUnpaidInvoiceID that returns the InvoiceID of the earliest invoice with an unpaid balance. Test the function in the following SELECT statement:

```
SELECT VendorName, InvoiceNumber, InvoiceDueDate,
       InvoiceTotal - CreditTotal - PaymentTotal AS Balance
FROM Vendors JOIN Invoices
  ON Vendors.VendorID = Invoices.VendorID
WHERE InvoiceID = dbo.fnUnpaidInvoiceID()
```

6. Create a table-valued function named fnDateRange, similar to the stored procedure of exercise 4. The function requires two parameters of data type smalldatetime. Don't validate the parameters. Return a result set that includes the InvoiceNumber, InvoiceDate, InvoiceTotal, and Balance for each invoice for which the InvoiceDate is within the date range. Invoke the function from within a SELECT statement to return those invoices with InvoiceDate between April 10 and April 20, 2002.
7. Use the function you created in exercise 6 in a SELECT statement that returns five columns: VendorName and the four columns returned by the function.

8. Create a trigger for the Invoices table that automatically inserts the vendor name and address for a paid invoice into a table named ShippingLabels. The trigger should fire any time the PaymentTotal column of the Invoices table is updated. The structure of the ShippingLabels table is as follows:

```
CREATE TABLE ShippingLabels
(VendorName       varchar(50),
 VendorAddress1   varchar(50),
 VendorAddress2   varchar(50),
 VendorCity       varchar(50),
 VendorState      char(2),
 VendorZipCode    varchar(20))
```

9. A column that accepts null values but has a unique constraint can only have a single row with a null value. Write a trigger that prohibits duplicates, except for nulls. Use the following table. If an INSERT or UPDATE statement creates a duplicate value in the NoDupName column, roll back the statement and return an error message.

```
CREATE TABLE TestUniqueNulls
(RowID      int  IDENTITY  NOT NULL,
 NoDupName  varchar(20)    NULL)
```

Chapter 15
How to work with cursors

Objectives

- Given a specification for a database problem that can be solved by using Transact-SQL cursors, write a script that solves it.
- Explain the two ways in which cursors are implemented for a SQL Server database.
- Describe the seven types of SQL Server cursors.
- Explain the terms *scrollable* and *sensitive* as they apply to cursors.
- Explain how to update or delete data through a cursor.
- Given a script, stored procedure, user-defined function, or trigger that uses cursors, explain what each statement does.

Summary

- A *cursor* is a database object that points to a result set. You use a cursor to identify the row you want to retrieve from the result set.
- Most application programs use standard *API cursors* to access the data in a SQL Server database. If not, you can use *Transact-SQL cursors* to provide database access. You can also use Transact-SQL cursors within your own scripts or procedures.
- SQL Server supports seven types of cursors that differ based on their *scrollability* and their *sensitivity*.
- You can use a *scrollable cursor* to retrieve the row before or after the current row. You can use a *forward-only cursor* only to return the row after the current row.
- A *dynamic cursor* is sensitive to all changes to the data source. A *keyset-driven cursor* is sensitive to updates and deletes to the data source. A *static cursor* is insensitive to any changes to the data source.
- To use a cursor, you define it using the DECLARE statement; you open and populate it using the OPEN statement; you retrieve rows using the FETCH statement; you close it using the CLOSE statement; and you delete the cursor definition and release the resources it uses using the DEALLOCATE statement.
- When you define a cursor, you specify its *scope*, scrollability, and sensitivity; the kind of lock that's placed on a row when data is updated through the cursor; and the columns in the cursor that can be updated.
- You can use the FETCH statement to retrieve the next row, the previous row, the first row, the last row, the row that's *n* rows from the beginning of the result set, or the row that's *n* rows before or after the current row.

- You can use the @@FETCH_STATUS system function to determine the status of the most recently executed FETCH statement. You'll typically use this function in a WHILE loop to determine when the end of the result set is reached.
- You can use the @@CURSOR_ROWS system function to get the number of rows in a result set.
- *Concurrency* occurs when two or more users are working with the same data at the same time. SQL Server can manage concurrency by placing a lock on a row while a user is working with it.
- With *pessimistic locking*, the system places a lock on a row when it's fetched. That way, no other user can modify or delete the row until the lock is released.
- With *optimistic locking*, no lock is placed on a row. Then, an error is raised if you try to modify or delete the row through the cursor and the row no longer exists or it has been changed by another user.
- To update or delete the last row that was fetched through a cursor, you code a WHERE CURRENT OF clause that names the cursor in the UPDATE or DELETE statement.

Terms

cursor
API (application programming interface)
API cursor
Transact-SQL cursor
cursor scrollability
cursor sensitivity
forward-only cursor
dynamic cursor
keyset-driven cursor
keyset
static cursor
scope
concurrency
lock
optimistic locking
pessimistic locking

Self-study questions

1. What type of SQL Server cursor implementation is typically used by application programs? How else can cursors be implemented?
2. What is a *scrollable cursor*? How does it compare to a *forward-only cursor*?
3. What does the *sensitivity* of a cursor indicate? What is the sensitivity of a *dynamic cursor*? a *keyset-driven cursor*? a *static cursor*?
4. What three additional types of forward-only cursors does SQL Server provide? How do they differ from the standard forward-only cursor?
5. What sequence of statements do you typically use to process a cursor?
6. What options do you have for fetching a row from a cursor?
7. How can you determine whether or not a fetch operation was successful?
8. How can you determine the number of rows in a cursor result set?

9. What is concurrency? What two types of locks can SQL Server use to manage concurrency, and how do they work?
10. How do you update or delete the last row that was retrieved through a cursor?

Exercises

1. Write a script to declare and use a cursor for the following SELECT statement. Use a WHILE loop to fetch each row in the result set. Omit the INTO clause to fetch directly to the Grids tab.

```
SELECT VendorName, AVG(InvoiceTotal) AS InvoiceAvg
FROM Vendors JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
GROUP BY VendorName
```

2. Modify the solution to exercise 1 to fetch each row into a set of local variables. Use the PRINT statement to return each row in the format "Name, $0.00" to the Messages tab.
3. Write a script that uses a cursor and dynamic SQL to output one row from each base user table in the AP database. Specifically exclude the table named "dtproperties" from the result set.
4. Write a script that uses a cursor for the following SELECT statement:

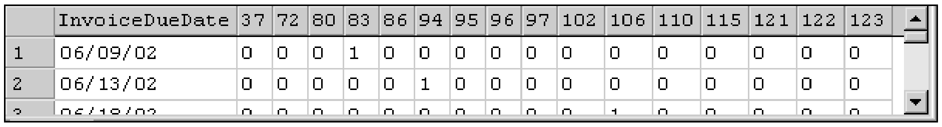
```
SELECT VendorName, InvoiceNumber, InvoiceAvg,
       (InvoiceTotal - InvoiceAvg) AS AmtOverAvg
FROM Vendors JOIN Invoices
    ON Vendors.VendorID = Invoices.VendorID
JOIN
    (SELECT VendorID, AVG(InvoiceTotal) AS InvoiceAvg
    FROM Invoices
    GROUP BY VendorID) AS AvgInvoices
    ON Vendors.VendorID = AvgInvoices.VendorID
WHERE (InvoiceTotal - InvoiceAvg) > 0
ORDER BY VendorName, InvoiceNumber
```

For each row in the result set, return two lines to the Messages tab in the following format:

```
Blue Cross, Inv# 547481328
  $36.00 over vendor average of $188.00.
```

Challenge project

5. Write a script that generates a cross-tabulation of VendorID by InvoiceDueDate for all invoices with a balance due. Return one column for each vendor with a balance due, similar to the example presented in figure 13-12 of the text. In addition, return a single column for the invoice due date. Each cell in the final cross-tabulation should represent the number of invoices with that VendorID that are due on that InvoiceDueDate.



	InvoiceDueDate	37	72	80	83	86	94	95	96	97	102	106	110	115	121	122	123
1	06/09/02	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	06/13/02	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	06/18/02	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Chapter 16
How to manage transactions and locking

Objectives

- Given a set of statements to be combined into a transaction, insert the Transact-SQL statements to explicitly begin, commit, or roll back the transaction, as appropriate.
- Describe how you can work with an implicit transaction that's created due to the autocommit mode of SQL Server.
- Describe the effect of the COMMIT TRAN statement on a nested transaction.
- Define the term *concurrency* and explain how locking prevents concurrency problems.
- Given a set of statements, identify which are most likely to cause concurrency problems.
- Identify the types of concurrency problems that can be prevented by locking.
- Describe the effect that the transaction isolation level has on locking and concurrency.
- Describe resource granularity and define the term *lock escalation*.
- Identify the two lock mode categories and define *lock promotion*.
- Describe how a deadlock occurs and what happens when SQL Server identifies a deadlock.
- Describe three coding techniques that can reduce deadlocks.
- Given a script, stored procedure, user-defined function, or trigger that uses explicit transactions, explain what each statement does.

Summary

- A *transaction* is a group of database operations that are combined into a logical unit.
- When you *commit* a transaction, the operations become a permanent part of the database. Until it's committed, you can undo the operations by *rolling back* the transaction.
- To start a transaction explicitly, you code the BEGIN TRAN statement. If you don't code this statement, each statement is treated as a separate transaction that is rolled back if an error occurs or is committed otherwise.
- You can *nest* a transaction within another transaction. However, the behavior of nested transactions is counterintuitive, and they're used infrequently.
- You can use the SAVE TRAN statement to create a save point. Then, you can code the name of the save point in a ROLLBACK TRAN statement to roll back all of the statements to that save point.

- *Concurrency* is the ability of a system to support two or more transactions working with the same data at the same time. Concurrency can be a problem when the data is being modified.
- You can avoid some concurrency problems by using *locks*, which delay the executing of a transaction if it conflicts with a transaction that's already running.
- You can set the *transaction isolation level* to control the degree to which transactions are isolated from one another. The server isolates transactions by using more restrictive locking behavior.
- SQL Server can lock data at various levels, from a single row to an entire database. These are known as *lockable resources*. The levels are based on *granularity*, where a *course-grain lock* affects more data than a *fine-grain lock*.
- The SQL Server *lock manager* automatically assigns locks for each transaction. It can also *escalate* finer-grain locks on the same resource to a single coarse-grain lock.
- The lock manager also assigns a *lock mode* to each transaction. In general, retrieval operations acquire *shared locks*, and update operations acquire *exclusive locks*. As a single transaction is being processed, its lock may have to be *promoted* from one lock mode to a more exclusive lock mode.
- An *intent lock* indicates that SQL Server intends to acquire a shared lock or an exclusive lock on a finer-grain resource. This prevents another transaction from acquiring conflicting locks.
- A *deadlock* occurs when two transactions are simultaneously holding and requesting a lock on each other's resource. SQL Server automatically detects deadlocks and rolls back one of the transactions. The transaction that's rolled back is called the *deadlock victim.*
- You can prevent deadlocks by using the lowest possible transaction isolation level, by not allowing transactions to remain open very long, by making large changes when you can be assured of nearly exclusive access, and by considering locking when you code your transactions.

Terms

transaction	granularity
commit a transaction	fine-grain lock
roll back a transaction	coarse-grain lock
autocommit mode	lock manager
nested transactions	lock escalation
save point	lock mode
concurrency	shared lock
locking	exclusive lock
lost update	update lock
dirty read	intent lock
nonrepeatable read	schema lock
phantom read	lock promotion
transaction isolation level	deadlock
lockable resource	deadlock victim

Self-study questions

1. In SQL Server's autocommit mode, what does an implicit transaction consist of? What kind of control do you have over this type of transaction?
2. What happens when you commit a transaction? What if the transaction is nested within another transaction?
3. What happens when you roll back a transaction? How can you partially roll back a transaction?
4. What is concurrency? How can locking prevent concurrency problems?
5. How can a lost update occur? a dirty read? a nonrepeatable read? a phantom read?
6. How does the transaction isolation level affect locking?
7. What is the granularity of a lockable resource? What happens when the SQL Server lock manager detects that several fine-grain locks apply to a single coarse-grain resource?
8. What are the two main categories of locks and when are they used?
9. What does lock promotion refer to and when does it occur?
10. How does a deadlock occur? How does SQL Server handle a deadlock? How can you reduce deadlocks?

Exercises

1. Write a set of action queries coded as a transaction to reflect the following change: United Parcel Service has been purchased by Federal Express Corporation and the new company is named FedUP. Rename one of the vendors and delete the other after updating the VendorID column in the Invoices table.
2. Write a set of action queries coded as a transaction to move rows from the Invoices table to the InvoiceArchive table. Insert all paid invoices from Invoices into InvoiceArchive, but only if the invoice doesn't already exist in the InvoiceArchive table. Then delete all paid invoices from the Invoices table, but only if the invoice exists in the InvoiceArchive table.

Chapter 17
How to manage database security

Objectives

- Given a specification for a new user's security permissions, write the Transact-SQL statements that create the new user and grant the security permissions.
- Given a specification for a new user's security permissions, use the Enterprise Manager to create the new user and grant the security permissions.
- Given a specification for a set of security permissions, write the Transact-SQL statements to create a new role.
- Given a specification for a set of security permissions, use the Enterprise Manager to create a new role.
- Identify the two ways that SQL Server can authenticate a login ID.
- Identify the two SQL Server authentication modes.
- Describe the difference between an object permission and a statement permission.
- Describe the difference between a denied permission and a revoked permission.
- Describe the difference between an application role and a standard database role.
- Given a specification for a new user's job function, identify appropriate security permissions.

Summary

- A network user typically logs on to the network using a *login ID* and a password. Then, the security configuration determines which database objects the user can work with and which SQL statements the user can execute.
- You can log on to SQL Server using either *Windows authentication mode* or *SQL Server authentication mode*. With Windows authentication, access to SQL Server is controlled by Windows security. With SQL Server authentication mode, access to SQL Server is controlled by SQL Server security.
- *Object permissions* determine what actions a user can take on a specific database object, and a *statement permission* determines whether or not a user can execute a specific SQL DDL statement.
- You can create collections of permissions called *roles*. Then, you can assign the permissions in the role to a user by assigning the user to the role.
- SQL Server has built-in, or *fixed*, roles defined at the server level and at the database level. *Fixed server roles* typically include users who manage the server, and *fixed database roles* typically include users who manage or use the database. You can also create *user-defined roles* that consist of the permissions you specify.

- An *application role* is a special type of database role that's activated for a connection to the database. Then, the normal security for the login ID that was used to open the connection is replaced by the security that's specified by the application role. Application roles are intended for use by applications that manage their own security.
- You can create collections of Windows users called *groups*. Then, you can assign permissions and roles to the entire group.
- You can use Transact-SQL statements and system stored procedures to set up and manage security. Alternatively, you can use the Enterprise Manager.

Terms

login ID
permissions
object permissions
statement permissions
role
group
authentication mode
Windows authentication
SQL Server authentication
fixed role
fixed server role
fixed database role
user-defined role
application role

Self-study questions

1. In what two ways can SQL Server authenticate a login ID?
2. What are the two SQL Server authentication modes? When are they typically used?
3. How do object permissions control a user's access to a database? What are the standard object permissions?
4. How do statement permissions control a user's access to a database? What are some of the statements that can be explicitly permitted?
5. What is the difference between a denied permission and a revoked permission?
6. What permissions might you grant to a typical application programmer?
7. What is a role? What are the fixed server and fixed database roles typically used for?
8. What fixed role might you assign a user to if they will be responsible for managing the security for the entire server? a specific database?
9. What's the difference between a standard database role and an application role?

Exercises

1. Write a script that creates a user-defined database role named PaymentEntry in the AP database. Give UPDATE permission to the new role for the Invoices table, UPDATE and INSERT permission for the InvoiceLineItems table, and SELECT permission for all user tables.
2. Write a script that (1) creates a login ID named "AAaron" with the password "aaar9999"; (2) gives the user access to the AP database; and (3) assigns the user to the PaymentEntry role you created in exercise 1.
3. Write a script that creates four login IDs based on the contents of a new table called NewLogins. Use dynamic SQL to perform three actions for each row in this table: (1) create a login with a temporary password based on the first four letters of the login name followed by "9999"; (2) give the user access to the AP database; and (3) assign the user to the PaymentEntry role you created in exercise 1.

```
CREATE TABLE NewLogins
(LoginName varchar(128))

INSERT NewLogins
VALUES ('BBrown')
INSERT NewLogins
VALUES ('CChaplin')
INSERT NewLogins
VALUES ('DDyer')
INSERT NewLogins
VALUES ('EEbbers')
```

4. Write a script that removes the user-defined database role named PaymentEntry.
5. Using the Enterprise Manager, recreate the user-defined database role named PaymentEntry, as described exercise 1.
6. Using the Enterprise Manager, create a login ID named "FFalk" with the password "ffal9999," then give the user access to the AP database and assign the user to the PaymentEntry role you created in exercise 5.