

CA-IDMS[®]

SQL Self-Training Guide
15.0



Computer Associates™

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

THIS DOCUMENTATION MAY NOT BE COPIED, TRANSFERRED, REPRODUCED, DISCLOSED, OR DUPLICATED, IN WHOLE OR IN PART, WITHOUT THE PRIOR WRITTEN CONSENT OF CA. THIS DOCUMENTATION IS PROPRIETARY INFORMATION OF CA AND PROTECTED BY THE COPYRIGHT LAWS OF THE UNITED STATES AND INTERNATIONAL TREATIES.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

THE USE OF ANY PRODUCT REFERENCED IN THIS DOCUMENTATION AND THIS DOCUMENTATION IS GOVERNED BY THE END USER'S APPLICABLE LICENSE AGREEMENT.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227.7013(c)(1)(ii) or applicable successor provisions.

First Edition, December 2000

© 2000 Computer Associates International, Inc.
One Computer Associates Plaza, Islandia, NY 11749
All rights reserved.

All trademarks, trade names, service marks, or logos referenced herein belong to their respective companies.

Contents

How to Use This Manual	vii
------------------------	-----

Part I. Introduction to Relational Databases and SQL

Chapter 1. Relational Database Concepts	1-1
1.1 About this chapter	1-3
1.2 Tables	1-4
1.3 Relationships among tables	1-6
1.4 Relational operations	1-8
1.5 Benefits of a relational database	1-10
1.6 Review	1-11
Chapter 2. What Is SQL?	2-1
2.1 About this chapter	2-3
2.2 Why SQL	2-4
2.2.1 What can SQL do?	2-4
2.3 Components of an SQL statement	2-6
2.4 Interactive and embedded SQL	2-7
2.5 Review	2-8

Part II. Using SQL Data Manipulation Language

Chapter 3. Retrieving Data	3-1
3.1 About this chapter	3-3
3.2 Retrieving all columns from a table	3-4
3.2.1 Exercise 3-1	3-5
3.3 Retrieving selected columns from a table	3-7
3.3.1 Exercise 3-2	3-8
3.3.2 Exercise 3-3	3-8
3.4 Renaming column headings	3-10
3.4.1 Exercise 3-4	3-11
3.5 Displaying calculations in columns	3-12
3.5.1 Exercise 3-5	3-13
3.6 Eliminating duplicate rows	3-21
3.6.1 Exercise 3-6	3-26
3.7 Organizing data	3-28
3.7.1 Exercise 3-7	3-30
3.7.2 Exercise 3-8	3-31
3.7.3 Exercise 3-9	3-33
3.7.4 Exercise 3-10	3-36
3.8 Review	3-38
3.9 Scenarios	3-39
Chapter 4. Using Conditional Retrieval	4-1
4.1 About this chapter	4-3

4.2	The WHERE clause	4-4
4.3	Comparison operators and keywords in predicates	4-5
4.4	Using comparison operators in predicates	4-6
4.4.1	Exercise 4-1	4-7
4.4.2	Exercise 4-2	4-8
4.4.3	Exercise 4-3	4-9
4.4.4	Exercise 4-4	4-9
4.5	Using keywords in predicates	4-11
4.5.1	Exercise 4-5	4-11
4.5.2	Exercise 4-6	4-13
4.5.3	Exercise 4-7	4-15
4.5.4	Exercise 4-8	4-17
4.5.5	Exercise 4-9	4-19
4.5.6	Exercise 4-10	4-19
4.6	Using calculated values in predicates	4-20
4.6.1	Exercise 4-11	4-21
4.7	Combining predicates	4-22
4.7.1	Exercise 4-12	4-22
4.8	Review	4-27
4.9	Scenarios	4-29
Chapter 5. Using Aggregate Functions		5-1
5.1	About this chapter	5-3
5.2	Aggregate functions	5-4
5.2.1	Exercise 5-1	5-5
5.2.2	Exercise 5-2	5-5
5.2.3	Exercise 5-3	5-6
5.2.4	Exercise 5-4	5-7
5.2.5	Exercise 5-5	5-8
5.2.6	Exercise 5-6	5-8
5.2.7	Exercise 5-7	5-9
5.3	Eliminating duplicate rows	5-10
5.3.1	Exercise 5-8	5-10
5.4	Grouping information	5-11
5.4.1	Exercise 5-9	5-12
5.4.2	Exercise 5-10	5-12
5.5	Using HAVING	5-15
5.5.1	Exercise 5-11	5-15
5.5.2	Exercise 5-12	5-16
5.6	Renaming column headings	5-17
5.6.1	Exercise 5-13	5-17
5.7	Review	5-18
5.8	Scenarios	5-19
Chapter 6. Accessing Multiple Tables		6-1
6.1	About this chapter	6-3
6.2	What is a join operation?	6-4
6.2.1	Joining tables on common columns	6-4
6.2.1.1	Exercise 6-1	6-6
6.2.2	Qualifying a column name	6-7
6.2.3	Qualifying a table name	6-9

6.2.3.1 Exercise 6-2	6-9
6.2.4 Sorting the result	6-10
6.2.4.1 Exercise 6-3	6-10
6.2.5 Additional search criteria in a join	6-12
6.2.6 Exercise 6-4	6-12
6.2.7 Things to remember about joining tables	6-12
6.3 Joining a table to itself	6-14
6.4 Using UNION	6-16
6.5 Review	6-18
6.6 Scenarios	6-19
Chapter 7. Nesting SELECT Statements	7-1
7.1 About this chapter	7-3
7.2 SELECT statement in a WHERE clause	7-4
7.3 Using a subquery with IN	7-5
7.3.1 Exercise 7-1	7-5
7.4 Using an aggregate function in a nested SELECT statement	7-7
7.4.1 Exercise 7-2	7-7
7.4.1.1 Exercise 7-2B	7-8
7.5 Using EXISTS	7-10
7.5.1 Exercise 7-3	7-11
7.6 Things to remember about subqueries	7-14
7.7 Review	7-15
7.8 Scenarios	7-16
Chapter 8. Updating a Table	8-1
8.1 About this chapter	8-3
8.2 Inserting data into a table	8-4
8.2.1 Exercise 8-1	8-4
8.2.2 Exercise 8-2	8-6
8.2.3 Exercise 8-3	8-6
8.3 Modifying data in a table with SET	8-8
8.3.1 Exercise 8-4	8-8
8.3.2 Exercise 8-5	8-10
8.3.3 Exercise 8-6	8-10
8.3.4 Exercise 8-7	8-11
8.4 Removing data from a table	8-12
8.4.1 Exercise 8-8	8-12
8.4.2 Exercise 8-9	8-13
8.5 Review	8-14

Part III. Appendixes

Appendix A. Sample Data Description Language	A-1
A.1 About this appendix	A-3
A.2 Table creation	A-4
A.3 Indexes	A-6
A.4 Views	A-7
A.5 Data integrity	A-8

Appendix B. Answers to Exercises	B-1
B.1 Chapter 1	B-3
B.2 Chapter 2	B-4
B.3 Chapter 3	B-5
B.4 Chapter 4	B-10
B.5 Chapter 5	B-14
B.6 Chapter 6	B-19
B.7 Chapter 7	B-24
B.8 Chapter 8	B-27
Appendix C. Table Descriptions	C-1
C.1 Table names and descriptions	C-3
Index	X-1

How to Use This Manual

What this guide is about

This self-training guide provides information on how to use interactive SQL data manipulation language (DML).

What you will learn

After reading this guide and doing the exercises, you should be able to:

- Describe a relational database and state its benefits
- Create SQL statements to retrieve data, based on specific criteria
- Create SQL statements to insert, modify, and delete data from a table

Who should use this guide

Anyone who will use basic SQL DML or who will use SQL in programs.

Online exercises

You can do the exercises in this guide **online** in any one of several processing environments. The exercises are designed to be used in the interactive environment.

If you want to do the exercises in this guide online, you must:

- Have online access to the demonstration database that is provided with the product installation and is used by the examples and exercises in this guide
- Know how to access and submit statement syntax to the interactive SQL tool in your environment
- Be familiar with the keyboard and terminal in your environment

Check with your system administrator for access to the appropriate system, database, and interactive SQL tool.

Accessing CA-IDMS/DB: Before you begin doing the exercises in this guide in the CA-IDMS/DB environment, be familiar with documentation of the tool you will use to submit SQL statements, such as the *CA-IDMS Command Facility* manual. Also, check with your system administrator to learn:

- The CA-IDMS/DC or CA-IDMS/UCF system to which you should sign on so that you can access the demonstration database online.

Tip: The exercises in this guide use mixed upper and lower case characters.

Before you invoke the interactive SQL tool, issue the DCUF SET UPLOW command to CA-IDMS.

- The dictionary to which your SQL session should be connected.
- The qualifiers of the demonstration database table names — a table name in an SQL statement must include the qualifier unless the qualifier matches the default schema for your SQL session.

Tip: You can set the default schema by submitting this statement: SET SESSION CURRENT SCHEMA *schema-name*.

- Whether you should roll back (eliminate) changes you make to the demonstration database with INSERT, UPDATE, and DELETE statements.

If so, submit this statement to the Command Facility before you begin the exercises:

```
set options autocommit off;
```

Then, after you finish a session of doing online exercises that update the database but before you exit the Command Facility, issue this statement:

```
rollback work;
```

How to proceed

If you have had no experience with relational databases, begin with Chapter 1, “Relational Database Concepts.” Read the chapters in order and do the exercises and review exercises in each chapter. Keep in mind that several people in your organization may use this guide, so you probably don't want to mark in it.

If you are familiar with relational database concepts, begin with Chapter 2, “What Is SQL?” and read the chapters in order.

How much time to allow: Allow five to eight hours to complete the entire self-training guide including the online practice exercises, assuming you are familiar with the keyboard. You can complete the self-training guide in one sitting or in several sessions as follows:

- Session 1 — Chapters 1 through 4
- Session 2 — Chapters 5 and 6
- Session 3 — Chapters 7 and 8

Practice exercises: Practice exercises begin in Chapter 3, “Retrieving Data.” Each exercise after the first builds on the previous exercise. If you are doing the exercises online, you can check your work by looking at the results shown after the exercise.

In Chapters 3 through 8, you see examples written out in full with the label **How it's done**. When you enter these statements online, you'll see a result table with the same contents as the one shown in the book. The table in the book may be abbreviated.

After each example and its result, there are exercises where the SQL statements are *not* given. Instead, a description of the requested information is given, and you write the statements necessary to achieve the result. These exercises are identified by the labels **Now you try it** and **Try another**.

Practicing without access to a database: You can go through these exercises without having access to a database. Simply write out your answers. Then check the correct answers in Appendix B, “Answers to Exercises” on page B-1.

Sample administrative statements: In Appendix A, “Sample Data Description Language” you will see sample statements for database definition that you *do not enter*. They are for your information only.

Chapter review: At the end of each chapter, you will find review exercises covering the material you have just studied. These exercises allow you to evaluate how well you have learned the material presented. You are encouraged to do them.

In addition, Chapters 3 through 7 include scenarios at the end. Each scenario requires you to create SQL statements to retrieve or update data based on a specific business requirement.

Answers to exercises: Answers to online exercises, reviews, and scenarios appear in Appendix B, “Answers to Exercises” on page B-1 B.

The demonstration database

In the online practice exercises, you will access data from the personnel database developed for a company called Commonwealth Auto.

Commonwealth Auto requires data to be maintained on all employees, jobs, skills, departments, benefits, and projects. Other associated employee information is also maintained, but you will not access it in these exercises.

The Human Resources and Accounting departments use the database for many of their activities. In this guide, these departments make requests for reports or information that you satisfy through your knowledge and use of SQL. The requests concern salary and budget information, department lists, and vacation and project updates. They range from the simple to the complex.

The requests are based on actual information maintained by a small corporation.

Database tables: The Commonwealth Auto database consists of two schemas:

- DEMOEMPL - tables containing employee information
- DEMOPROJ - tables containing project-related information

The information is maintained in several tables in the database. These are the tables in the portion of the database you will use:

Table	Schema	Contents
ASSIGNMENT	DEMOPROJ	The assignment of employees to projects
BENEFITS	DEMOEMPL	The benefits an employee has with the company
CONSULTANT	DEMOPROJ	Each consultant associated with the company
COVERAGE	DEMOEMPL	Employee's insurance information
DEPARTMENT	DEMOEMPL	Each department within the company
DIVISION	DEMOEMPL	Each division within the company
EMPLOYEE	DEMOEMPL	Personal information on each employee working for the company
EXPERTISE	DEMOPROJ	The skills each employee possesses
INSURANCE_PLAN	DEMOEMPL	Details of each insurance plan
JOB	DEMOEMPL	The jobs within the company
POSITION 1	DEMOEMPL	The jobs an employee has held and is currently holding within the company
PROJECT	DEMOPROJ	The projects within the company

Table	Schema	Contents
SKILL	DEMOPROJ	The skills throughout the company

Note: 1 - POSITION is also an SQL keyword; when it is used to qualify a column name, the table name must be enclosed in double quotation marks. For example, "POSITION".column-name. For information about qualifying column names, refer to 6.2.2, "Qualifying a column name" on page 6-7.

Appendix C, "Table Descriptions" presents a description of each column in each table in the database.

Part I. Introduction to Relational Databases and SQL

Chapter 1. Relational Database Concepts

- 1.1 About this chapter 1-3
- 1.2 Tables 1-4
- 1.3 Relationships among tables 1-6
- 1.4 Relational operations 1-8
- 1.5 Benefits of a relational database 1-10
- 1.6 Review 1-11

1.1 About this chapter

Goal: At the end of this chapter, you will be able to:

- Define basic relational database terms
- List the components of a relational database

Summary: A relational database is a collection of tables containing data. A table consists of columns (attributes that describe the table) and rows (actual occurrences of data). Data can be accessed easily and quickly in a relational database and is viewed in a tabular format.

1.2 Tables

Relational databases present information as a collection of tables. Unless empty, each table contains related data.

Sample tables: This diagram shows the EMPLOYEE, SKILL, DEPARTMENT, and PROJECT tables from the database for Commonwealth Auto:

EMP_ID	EMP_LNAME	EMP_FNAME	DEPT_ID
2096	CARLSON	THOMAS	4600
2437	THOMPSON	HENRY	4600
2598	JACOBS	MARY	5100

DEPT_ID	DEPT_NAME
5200	CORPORATE MARKETING
4600	MAINTENANCE
5100	BILLING

SKILL_ID	SKILL_NAME
4250	DATA ENTRY
4370	FILING
4490	GENERAL LEDGER

PROJ_ID	PROJ_DESC
C200	NEW BRAND RESEARCH
C240	SERVICE STUDY
D880	SYSTEM ANALYSIS

The EMPLOYEE table contains data about employees. The SKILL table contains information about skills that are used in Commonwealth Auto. The DEPARTMENT table contains information about the departments in the company. The PROJECT table contains information about projects.

A table is made up of columns and rows. A portion of the EMPLOYEE table in the Commonwealth Auto database looks like this:

Columns		
EMP_ID	EMP_LNAME	EMP_FNAME
2096	CARLSON	THOMAS
2437	THOMPSON	HENRY
2598	JACOBS	MARY

Rows

Columns: A table has one or more columns. Each column:

- Has entries containing a single type of data
- Is displayed vertically
- Is identified by a name

For example, the employee ID (EMP_ID) column contains employee IDs, each of which is a number. The employee IDs are listed one below the other. At the top of the column is a heading based on the kind of data in the column.

Rows: A table has zero or more rows. Each row:

- Contains one value in each column
- Is displayed horizontally
- Is not named

The first part of one row from the EMPLOYEE table looks like this:

2096	CARLSON	THOMAS
------	---------	--------

Primary keys: A business often needs to prevent duplicate rows of data from being stored in the same table. For example, each employee in the company needs an employee ID different from all other IDs. This is a way of distinguishing two employees who have the same name. You do not want to store two employees who have the same employee ID.

To ensure that duplicate rows are not stored, a column or combination of columns is identified as a **primary key** of the table when the table is defined. Each entry in the primary key column or columns must be unique; there can be no duplicates. As a result, the primary key uniquely identifies each row in the table.

A row of employee information in the EMPLOYEE table is uniquely identified by the employee ID. There is only one row with employee ID 2096 and only one row with employee ID 2437. However, there can be more than one employee with a first name of Mary. The column containing the first name is not a unique key:

EMP_ID	EMP_LNAME	EMP_FNAME
2096	CARLSON	THOMAS
2437	THOMPSON	HENRY
2598	JACOBS	MARY

↑
Primary key

When you request data from a table and specify a value for the primary key, you see only one row returned.

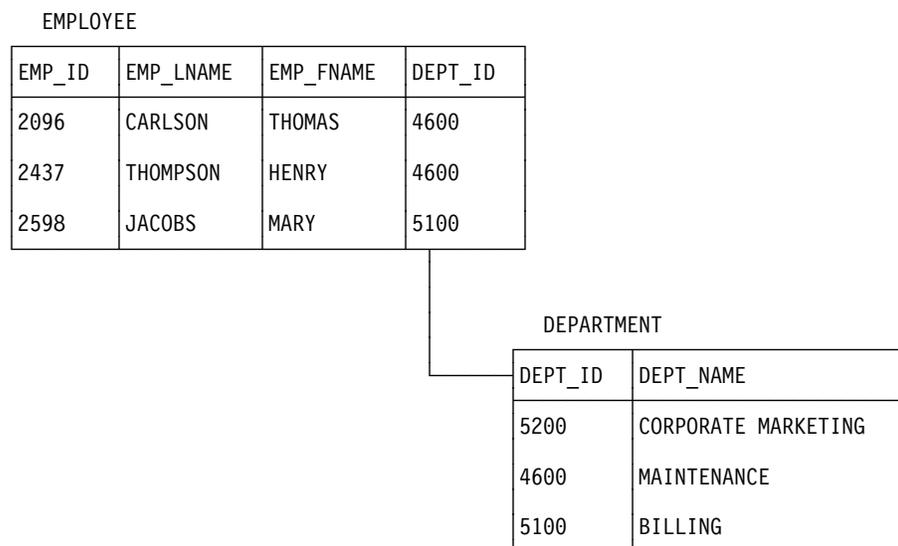
1.3 Relationships among tables

Normally, a database contains many tables holding related information. For example, in the Commonwealth Auto database, there is a table storing employee information and a table storing department information. Since each employee is associated with a department, there is a logical relationship between the two tables.

Foreign keys: The database designer establishes relationships among tables by defining **foreign keys**. A foreign key is a value or combination of values in a table that exists as the primary key in another table. The names of the columns that make up the foreign key do not have to be the same as the primary key column names.

When you need to retrieve data in two tables at the same time, you use a foreign key and a primary key as common columns (columns that are common between the tables).

Here's an illustration of the relationship between the EMPLOYEE and DEPARTMENT tables:



The department ID, DEPT_ID, is the primary key in the DEPARTMENT table and a foreign key in the EMPLOYEE table.

To find the name of the department that an employee is associated with, you would match the two tables based on this common column.

To find the name of the department that employee 2096 is associated with, you would look up the employee in the EMPLOYEE table based on the employee ID, 2096, and find department ID 4600. Then you would find the matching department ID 4600 in the DEPARTMENT table to find the department name, Maintenance.

1.4 Relational operations

You can manipulate tables to form new tables with relational operations.

The three types of operations that you use most often against a relational database involve accessing specified rows, particular columns, and more than one table.

Specified rows (SELECT): You can request that specific rows of data be retrieved from a table or tables.

For example, you can retrieve all information on employees whose last names are Carlson or Jacobs. Information on other employees is not returned. This type of operation is called a **select** operation.

EMPLOYEE

EMP_ID	EMP_LNAME	EMP_FNAME	DEPT_ID
2096	CARLSON	THOMAS	4600
2437	THOMPSON	HENRY	4600
2598	JACOBS	MARY	5100

EMP_ID	EMP_LNAME	EMP_FNAME	DEPT_ID
2096	CARLSON	THOMAS	4600
2598	JACOBS	MARY	5100

Particular columns (PROJECT): You can identify particular columns of data to be retrieved.

For example, you can retrieve only the last name and first name of each employee in the company, in order to create a personnel list. This type of operation is called a **project** operation.

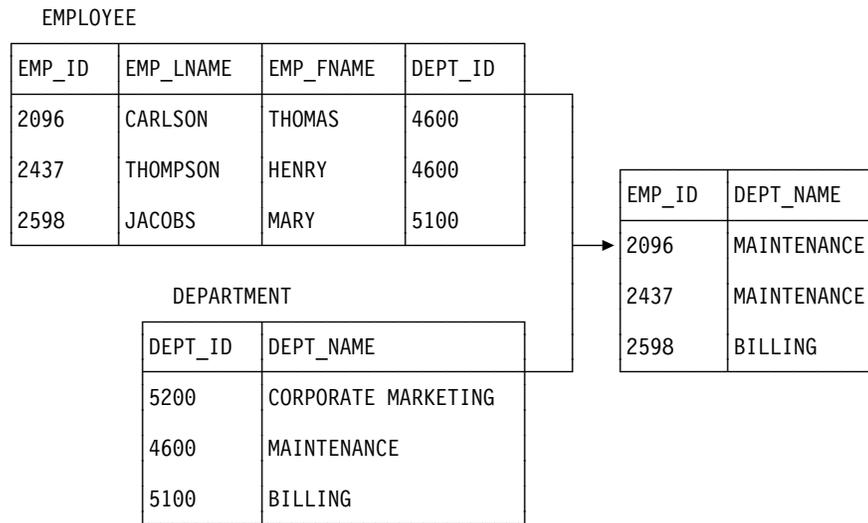
EMP_ID	EMP_LNAME	EMP_FNAME	DEPT_ID
2096	CARLSON	THOMAS	4600
2437	THOMPSON	HENRY	4600
2598	JACOBS	MARY	5100

EMP_LNAME	EMP_FNAME
CARLSON	THOMAS
THOMPSON	HENRY
JACOBS	MARY

More than one table (JOIN): You can retrieve data from more than one table at the same time.

For example, to create a list of department names and employees in each department, you need to retrieve information on each employee along with information on the department in which the employee works.

This data is in two tables: the employee information and department ID are in the EMPLOYEE table, and the department ID and department name are in the DEPARTMENT table. You can join the two tables to see both the employee and department information as a single table. This type of operation is called a **join** operation:



You use one or more of these basic operations to retrieve data from the database. For example, you may want to access two tables to see employee and department information (join) but show only the employee last name and the department name (project).

1.5 Benefits of a relational database

- You can access data in tables easily and quickly.
- It's easy to understand the data you see because it is in a tabular format.
- Relational databases are becoming standard on various computers.
- When you are designing application programs to access the database, you don't have to be aware of all the details of underlying physical database structures.
- You can make changes to the database without affecting application programs.

1.6 Review

Match each description on the left with a term or terms on the right. Terms can match more than one description.

Description	Term
1. Components of a relational database that hold the data	a. Foreign key
2. Components of a table	b. Primary key
3. A column or combination of columns holding values that form the primary key of another table	c. Tables
4. The types of operations you can perform against a relational database	d. Rows and columns
5. A way to establish a relationship between two tables	e. Select, project, and join
6. A column or combination of columns that uniquely identifies a row in a table	

To check your answers, see “Review answers” on page B-3

Chapter 2. What Is SQL?

- 2.1 About this chapter 2-3
- 2.2 Why SQL 2-4
 - 2.2.1 What can SQL do? 2-4
- 2.3 Components of an SQL statement 2-6
- 2.4 Interactive and embedded SQL 2-7
- 2.5 Review 2-8

2.1 About this chapter

Goal: At the end of this chapter, you will be able to:

- Define the term 'SQL'
- Specify why SQL is used and what you can do with it
- Identify the components of an SQL statement
- Compare interactive and embedded SQL

Summary: Structured Query Language (SQL) is a standardized non-procedural language used to retrieve and update information in a relational database.

2.2 Why SQL

SQL serves as a standard language that:

- Can be used either for ad hoc queries and updates or in application programs.
- Eliminates the need for the user to know how the database is physically structured.
- Facilitates the exchange of information from computer to computer and from database to database.

Benefits of a standard language

- You need less training when you move from one computer or product to another.
- One database management system can communicate with another if they use a standard interface.

2.2.1 What can SQL do?

You can use SQL to:

- Define a database
- Manipulate data in the database
- Control access to data in the database in a multi-user environment

Data definition: You use SQL data description language (DDL) statements to define a database and tables within the database.

Data manipulation: You use SQL DML statements to manipulate the data in tables.

There are four basic SQL DML statements:

- SELECT
- INSERT
- UPDATE
- DELETE

The SELECT statement is used to retrieve data. The result of a query is a **result table**. INSERT, UPDATE, and DELETE (all called update operations) are used to make changes to the data.

Data control: You use SQL DDL to control access to data in a multi-user environment. There are two basic SQL DDL commands:

- GRANT
- REVOKE

The GRANT command allows another user to access data and the REVOKE command removes that access.

If you cannot access a table, it probably means that you have not been granted access to it.

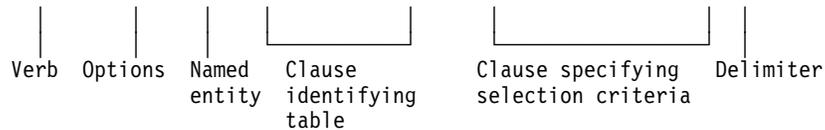
2.3 Components of an SQL statement

An interactive SQL statement consists of a structured set of English-like elements:

- A **verb** that tells the action you want performed.
- Additional **options** that modify verbs and further define the operation.
- **Named entities** that identify the object of the action.
- **Clauses** (required or optional) to identify the table in which the data is located and to specify more about how you want the action performed.
- A **delimiter** (;) that signals the end of the statement.

Basic SQL statement:

select distinct city from employee where emp_id > 5555;



A delimiter is required for interactive SQL commands. If you enter SQL commands in an application program, you may need to use a different delimiter or none at all.

Statement length: A statement can span several lines. It will not be executed until the delimiter is encountered.

You can issue only one interactive SQL statement at a time.

2.4 Interactive and embedded SQL

You can issue SQL statements either interactively or from within an application program.

Interactive SQL: When you use interactive SQL to enter a request or to change data, you get immediate results. This is the typical way of entering ad hoc statements.

For example, you might want to identify all employees who live in Boston. This SQL statement would return a table that includes the last name and first name of all employees residing in Boston:

```
select emp_lname, emp_fname
       from employee
       where city = 'Boston';
```

Embedded SQL: You can embed SQL statements in host application programs. With embedded SQL, the program receives the result of the request and acts on it, displays it, or prints it. For example, this embedded SQL statement returns to a COBOL program the last name and first name for employees living in the city requested by the program:

```
exec sql
select emp_lname, emp_fname
       into :emp_lname, :emp_fname
       from employee
       where city = :city_in
end-exec
```

2.5 Review

Fill in the blanks with the appropriate term or response:

1. SQL stands for _____ .
2. You use SQL _____ statements to define tables.
3. You use SQL _____ statements to change data in a table.
4. The three SQL update operations are _____, _____, and _____.
5. An interactive SQL statement ends with a _____.
6. An SQL statement begins with a _____.
7. An interactive SQL statement _____ (*can/cannot*) span several lines.
8. You _____ (*can/cannot*) issue several interactive SQL statements at once.
9. Interactive SQL gives you _____ results.
10. Embedded SQL returns the results to the _____.

To check your answers, see “Review answers” on page B-4

Part II. Using SQL Data Manipulation Language

Chapter 3. Retrieving Data

3.1 About this chapter	3-3
3.2 Retrieving all columns from a table	3-4
3.2.1 Exercise 3-1	3-5
3.3 Retrieving selected columns from a table	3-7
3.3.1 Exercise 3-2	3-8
3.3.2 Exercise 3-3	3-8
3.4 Renaming column headings	3-10
3.4.1 Exercise 3-4	3-11
3.5 Displaying calculations in columns	3-12
3.5.1 Exercise 3-5	3-13
3.6 Eliminating duplicate rows	3-21
3.6.1 Exercise 3-6	3-26
3.7 Organizing data	3-28
3.7.1 Exercise 3-7	3-30
3.7.2 Exercise 3-8	3-31
3.7.3 Exercise 3-9	3-33
3.7.4 Exercise 3-10	3-36
3.8 Review	3-38
3.9 Scenarios	3-39

3.1 About this chapter

Goal: After completing this chapter, you will be able to respond to requests for information from Commonwealth Auto by creating SQL statements that:

- Retrieve data from all columns and rows in a table.
- Retrieve data from specified columns in a table.
- Give new names to column headings.
- Display the results of calculations.
- Eliminate duplicate rows from your results.
- Sort the information displayed.

Summary: To retrieve data from the database, you use the SELECT statement, probably the most frequently used SQL statement.

Online exercises: The online exercises for this self-training guide begin in this chapter.

Important: Before you begin, be sure to read “Online exercises” on page xi in the preface of this guide.

When you see a complete statement ending with a delimiter (;) and the label **How it's done**, you can enter the statement online. The result you obtain should have the same content as the one in the book.

In most cases, after you have entered a statement, you will see another suggested retrieval with the label **Now you try it**. This time, you will use the knowledge you just gained to create your own statement *online*.

If your result does not match the one shown in the guide, or if you are unable to compose a statement, you can look up the correct syntax in Appendix B, “Answers to Exercises.”

At the end of this chapter, there are additional scenarios and review exercises to give you extra practice with SQL.

The database you'll use: You will use the Commonwealth Auto database for these online exercises. The tables in this database contain data about employees, jobs, skills, projects, and departments.

Before you begin the online exercises in this chapter, look in Appendix C, “Table Descriptions” at the type of information kept in each table.

3.2 Retrieving all columns from a table

The basic statement for retrieving data from a table is `SELECT`. `SELECT` specifies which data you want to retrieve. The `FROM` clause in the `SELECT` statement specifies which table holds the data.

How it's done: The `DEPARTMENT` table contains the following columns:

- `DEPT_ID`
- `DEPT_HEAD_ID`
- `DIV_CODE`
- `DEPT_NAME`

In order to list all information about each department, you need to access this table and select all columns. To do this, enter:

```
select *  
      from department;
```

You can enter this statement all on one line or spanning several lines. You can use either lowercase or uppercase.

What does the asterisk (*) mean?

It means that you want to see all the columns in the table. You don't have to list the column names explicitly.

What does `DEPARTMENT` indicate?

It's the name of the table from which you want to access data.

Why is there a semicolon at the end of the statement?

SQL will not process an interactive statement until it encounters a semicolon.

What you see: The result looks like this:

```

OCF 15.0 ONLINE IDMS NO ERRORS
SELECT * FROM DEPARTMENT;
**
** DEPT_ID  DEPT_HEAD_ID  DIV_CODE  DEPT_NAME
** -----  -
**    1120         2004  D06      PURCHASING - SERVICE
**    4200         1003  D04      LEASING - NEW CARS
**    4900         2466  D09      MIS
**    2210         2010  D04      SALES - NEW CARS
**    3520         3769  D04      APPRAISAL NEW CARS
**    5000         2466  D09      CORPORATE ACCOUNTING
**    4500         3222  D09      HUMAN RESOURCES
**    4600         2096  D06      MAINTENANCE
**    2200         2180  D02      SALES - USED CARS
**    5100         2598  D06      BILLING
**    6200         2461  D09      CORPORATE ADMINISTRATION
**    3530         2209  D06      APPRAISAL - SERVICE
**    6000         1003  D09      LEGAL
**    3510         3082  D02      APPRAISAL - USED CARS
**    1100         2246  D02      PURCHASING - USED CARS
**
** 17 rows processed

```

Rows are not ordered: There is no inherent order to the rows as they are stored in the database. The rows in your result, therefore, may be in a different order from those displayed here. The message specifying the number of rows returned may be worded differently and appear in a different position on your screen.

3.2.1 Exercise 3-1

Now you try it: Commonwealth Auto maintains information on all the skills the company requires to do business. This information is maintained in the `SKILL` table.

Enter a statement to access all skill information. It isn't important whether you use uppercase or lowercase in your SQL statement.

What table do you need to access?

You need to access the `SKILL` table.

The result looks like this:

3.2 Retrieving all columns from a table

SKILL_ID	SKILL_NAME	SKILL_DESC
5420	Writing	General writing skills
4444	Assembly	Auto body assembly experience
5160	Calculus	Knowledge of advanced mathematics
1000	Management	Experience managing people
4420	Telephone	Basic customer support
7000	Sales	Background in sales techniques
4410	Typing	Minimum 60 wpm
6666	Billing	Basic billing procedures
3065	Electronics	Electronic diagnosis and repair
5430	Mktng Writing	Background in promotional writing
6470	Window Installation	Installation of automotive windows
5130	Basic Math	Knowledge of basic math functions
5500	Gen Mktng	Knowledge of basic marketing concepts
5180	Statistics	Creating & evaluating statistics
6670	Gas Engine Repair	Experience in gasoline engine repair
6770	Purchasing	Basic buying & negotiation procedures
4370	Filing	Ability to organize correspondence/invoices
1030	Acct Mgt	Experience in managing acctng activities
5309	Appraising	Used car evaluation
4490	Gen Ledger	Experience with general ledger
6650	Diesel Engine Repair	Experience in diesel engine repair
4430	Interviewing	Basic interviewing experience
3333	Bodywork	Experience in repairing auto bodies
3088	Brake work	Brake diagnosis and repair
5200	Gen Acctng	Familiarity with basic AR and AP
4250	Data Entry	Familiarity with computer keyboard

26 rows processed

If your results do not match what you see above, check “Exercise 3-1 answer” on page B-5 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

3.3 Retrieving selected columns from a table

You just retrieved all columns and all rows from a table.

If you want to see only some of the columns, specify the names of the columns you want to see. Put a comma between column names.

How it's done: If you want to see only the department ID and name for each department in the company, enter:

```
select dept_id, dept_name
       from department;
```

The result of this SELECT statement is a list of the values in the DEPT_ID and DEPT_NAME columns.

Where did you get the column names?

They come from the table descriptions. In this guide, the table descriptions appear in Appendix C, "Table Descriptions."

The result looks like this:

```
DEPT_ID  DEPT_NAME
-----  -
1120    PURCHASING - SERVICE
4200    LEASING - NEW CARS
4900    MIS
2210    SALES - NEW CARS
3520    APPRAISAL NEW CARS
5000    CORPORATE ACCOUNTING
4500    HUMAN RESOURCES
4600    MAINTENANCE
2200    SALES - USED CARS
5100    BILLING
6200    CORPORATE ADMINISTRATION
3530    APPRAISAL - SERVICE
6000    LEGAL
3510    APPRAISAL - USED CARS
1100    PURCHASING - USED CARS
5200    CORPORATE MARKETING
1110    PURCHASING - NEW CARS
17 rows processed
```

How does this compare with the results displayed when you specified SELECT *?

You see only the columns you selected rather than all columns.

What determines the order of the columns?

It's based on the order in which you listed the columns in your SELECT statement.

3.3.1 Exercise 3-2

Now you try it: The Human Resources department is responsible for keeping track of all skill information for the company. Right now, they need to see which skill IDs and names are currently on file. Create a `SELECT` statement to retrieve and display skill IDs and names from the `SKILL` table.

Look in Appendix C, “Table Descriptions” to identify the column names.

The result looks like this:

```
SKILL_ID SKILL_NAME
-----
5420 Writing
4444 Assembly
5160 Calculus
1000 Management
4420 Telephone
7000 Sales
4410 Typing
6666 Billing
3065 Electronics
5430 Mktng Writing
6470 Window Installation
5130 Basic Math
5500 Gen Mktng
5180 Statistics
6670 Gas Engine Repair
6770 Purchasing
4370 Filing
1030 Acct Mgt
5309 Appraising
4490 Gen Ledger
6650 Diesel Engine Repair
4430 Interviewing
3333 Bodywork
3088 Brake work
5200 Gen Acctng
4250 Data Entry
```

If your results do not match what you see above, check “Exercise 3-2 answer” on page B-5 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

What message do you see?

You see a message stating the number of rows that have been retrieved.

3.3.2 Exercise 3-3

Try another: The president of the company wants a Christmas card list of all employees. He wants to see the first and last names for each employee and the street and community in which they live.

Enter a `SELECT` statement to do this, using the `EMPLOYEE` table.

The result looks like this:

EMP_FNAME	EMP_LNAME	STREET	CITY
Samuel	Spade	47 London St	Canton
Catherine	Williams	566 Lincoln St	Boston
Janice	Dexter	399 Pine St	Medford
Cora	Parker	2 Spring St	Boston
Mark	White	560 Camden St	Canton
Marylou	Hamel	11 Main St	Medford
James	Gallway	12 East Speen St	Stoneham
Ronald	Wilder	30 Heron Ave	Natick
Frank	Lowe	25 Rutland St	Natick
Mary	Umidy	895A Braintree Circle	Medford
Cynthia	Taylor	201 Washington St	Concord
John	Brooks	129 Bedford St	Camden
Carl	Smith	18 South St	Newton
Martin	Loren	401 Cross St	Grover
Bruce	MacGregor	254 Waterside Rd	Camden
Michael	Smith	201 Summer St	Brookline
William	Griffin	390 Sherman St	Taunton
Jason	Thompson	3 Flintlock St	Natick
Stephen	Wills	34 Avon Dr	Canton
David	Alexander	18 Cross St	Grover

55 rows processed

If your results do not match what you see above, check “Exercise 3-3 answer” on page B-5 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

3.4 Renaming column headings

In your results, each column has a heading. The heading is the name of the column as it was specified in the table definition.

EMP_FNAME	EMP_LNAME	STREET	CITY
Samuel	Spade	47 London St	Canton
Catherine	Williams	566 Lincoln St	Boston
Janice	Dexter	399 Pine St	Medford
Cora	Parker	2 Spring St	Boston
Mark	White	560 Camden St	Canton
		.	
		.	
		.	
55 rows processed			

How it's done: To make a column heading more meaningful, you can rename it. To do this, add AS and the name you want to use after each column name:

```
select emp_fname as "First Name",
       emp_lname as "Last Name",
       street as "Street",
       city as "City"
from employee;
```

Enclose new heading names in double quotation marks.

Don't forget the commas between column names.

This is the same SELECT statement you used to retrieve selected columns. However, in this case, the headings will have a more meaningful appearance.

These new column headings assigned using AS are temporary names and appear only in the display.

The result looks like this:

```

First Name      Last Name      Street          City
-----
Samuel          Spade          47 London St   Canton
Catherine      Williams      566 Lincoln St Boston
Janice         Dexter        399 Pine St    Medford
Cora           Parker        2 Spring St    Boston
Mark           White         560 Camden St  Canton
Marylou        Hamel         11 Main St     Medford
James          Gallway       12 East Speen St Stoneham
Ronald         Wilder        30 Heron Ave   Natick
Frank          Lowe          25 Rutland St  Natick
Mary           Umidy         895A Braintree Circle Medford
Cynthia        Taylor        201 Washington St Concord
John           Brooks        129 Bedford St Camden
Carl           Smith         18 South St    Newton
Martin         Loren         401 Cross St   Grover
Bruce          MacGregor     254 Waterside Rd Camden
Michael        Smith         201 Summer St  Brookline
William        Griffin       390 Sherman St Taunton
Jason          Thompson      3 Flintlock St Natick
Stephen        Wills        34 Avon Dr     Canton
David          Alexander     18 Cross St    Grover
                .
                .
                .
55 rows processed

```

3.4.1 Exercise 3-4

Now you try it: Earlier, you produced a report that displayed department ID and name from the DEPARTMENT table. Produce the same report and rename the headings "Department ID" and "Name." The result looks like this:

```

DEPARTMENT ID  NAME
-----
1120 PURCHASING - SERVICE
4200 LEASING - NEW CARS
4900 MIS
2210 SALES - NEW CARS
3520 APPRAISAL NEW CARS
5000 CORPORATE ACCOUNTING
4500 HUMAN RESOURCES
4600 MAINTENANCE
2200 SALES - USED CARS
5100 BILLING
6200 CORPORATE ADMINISTRATION
3530 APPRAISAL - SERVICE
6000 LEGAL
3510 APPRAISAL - USED CARS
1100 PURCHASING - USED CARS
5200 CORPORATE MARKETING
1110 PURCHASING - NEW CARS
17 rows processed

```

If your results do not match what you see above, check "Exercise 3-4 answer" on page B-5 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

3.5 Displaying calculations in columns

You can use arithmetic expressions to calculate new values from a column.

Use the following symbols for arithmetic operations:

Symbol	Meaning
*	Multiplication
/	Division
+	Addition
-	Subtraction

Order of evaluation: Multiplication and division are performed first, from left to right. Addition and subtraction are performed second, from left to right. You can control the order in which operations are performed by using parentheses to enclose the operations you want performed first.

Computing with null values: Unless the column definition specifies otherwise, a column can contain no value. No value is also called **null**, or a **null value**. The result table usually shows null values as '<null>'.

The result of any calculation involving a null value is always a null value.

How it's done: This year, the base rate for all jobs is rising 5 percent above last year's rates. The budget group needs a report showing job ID, last year's rate, and last year's rate plus 5 percent. This information is contained in the JOB table. To display the new rate, you will have to multiply the current rate by 1.05.

To create a table showing job ID, last year's rate, and this year's rate, enter:

```
select job_id as "Job", min_rate as "Minimum Rate",  
       min_rate * 1.05 as "Adjusted Rate"  
from job;
```

You can omit the space on either side of the arithmetic symbol.

The result looks like this:

JOB	MINIMUM RATE	ADJUSTED RATE
---	-----	-----
8001	90000.00	94500.0000
2077	17000.00	17850.0000
9001	111000.00	116550.0000
3051	8.50	8.9250
4700	33000.00	34650.0000
3029	25000.00	26250.0000
6011	59400.00	62370.0000
4130	35000.00	36750.0000
4666	41000.00	43050.0000
4123	35000.00	36750.0000
5555	30000.00	31500.0000
4025	31000.00	32550.0000
4023	44000.00	46200.0000
2051	8.80	9.2400
4734	25000.00	26250.0000
5110	40000.00	42000.0000
2053	8.80	9.2400
6004	66000.00	69300.0000
5111	27000.00	28350.0000
4012	21000.00	22050.0000
2055	17000.00	17850.0000
4560	11.45	12.0225
5890	45000.00	47250.0000
3333	21600.00	22680.0000
6021	76000.00	79800.0000
25 rows processed		

Why did you provide a heading for the calculated column?

You provided a heading to have a more meaningful name. If you hadn't, the heading would have been **(EXPR)** or **Expression**.

3.5.1 Exercise 3-5

Now you try it: The Corporate Marketing department is considering revamping the bonus system. They want a report showing employee IDs, how much salary they earned and, if any, the bonus percentage and how much bonus each employee earned. This information is maintained in the POSITION table as SALARY_AMOUNT and BONUS_PERCENT.

Enter a SELECT statement to display this information. Rename the columns appropriately.

The result looks like this:

3.5 Displaying calculations in columns

EMPLOYEE	SALARY	BONUS PERCENTAGE	BONUS PAID
3411	53665.00	<null>	<null>
3411	44001.40	<null>	<null>
4773	45240.00	<null>	<null>
2010	76440.00	0.275	21021.00000
3338	22048.84	<null>	<null>
2246	59488.00	<null>	<null>
2246	29536.00	<null>	<null>
1034	<null>	<null>	<null>
2424	<null>	<null>	<null>
3767	50440.50	0.230	11601.31500
3767	2200.00	<null>	<null>
3449	74776.00	<null>	<null>
3082	68016.00	<null>	<null>
3341	48465.80	<null>	<null>
4660	36400.00	0.250	9100.00000
4660	24000.00	<null>	<null>
2209	66144.00	<null>	<null>
2894	111593.00	<null>	<null>
4001	36921.00	0.230	8491.83000
5090	48568.48	0.205	9956.53840
1765	47009.34	<null>	<null>
4456	<null>	<null>	<null>
1765	18001.00	<null>	<null>
3991	42016.00	0.235	9873.76000
3991	27976.00	<null>	<null>
3778	<null>	<null>	<null>
4358	57824.50	<null>	<null>
4962	30680.00	<null>	<null>
2180	76961.00	<null>	<null>
2180	19000.10	<null>	<null>
2106	23920.00	<null>	<null>
3222	110448.00	<null>	<null>
4002	28601.80	<null>	<null>
2437	<null>	<null>	<null>
2096	85280.00	<null>	<null>
2096	<null>	<null>	<null>
2004	59280.00	<null>	<null>
2004	<null>	<null>	<null>
5103	<null>	<null>	<null>
5008	47944.00	<null>	<null>
4321	56977.80	<null>	<null>
2598	<null>	<null>	<null>
3764	54184.00	0.260	14087.84000
3764	28912.00	<null>	<null>
2448	70720.00	0.255	18033.60000
2461	43784.00	<null>	<null>
1234	117832.68	<null>	<null>
1003	146432.00	<null>	<null>
4027	28081.40	<null>	<null>
2466	94953.52	<null>	<null>
2174	49921.76	<null>	<null>
2781	43888.00	<null>	<null>
3704	22880.00	<null>	<null>
4008	24441.00	<null>	<null>
3433	<null>	<null>	<null>
3288	<null>	<null>	<null>
3841	33800.00	<null>	<null>
4703	24857.00	<null>	<null>
3294	53665.56	<null>	<null>
3769	41600.00	<null>	<null>
3118	45241.94	<null>	<null>

61 rows processed

If your results do not match what you see above, check “Exercise 3-5 answer” on page B-5 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

Why the '<null>' entries?

Most positions are not sales positions and do not have bonuses attached.

Using parentheses: Use parentheses to specify the order in which you want the arithmetic evaluation to take place.

How it's done: You have been asked to produce a report that shows what weekly salaries would look like before and after a raise of \$1,000 per year. Show yearly salary as well.

Try this without using parentheses first:

```
select salary_amount,  
       salary_amount / 52 as "Current Wkly Sal",  
       salary_amount + 1000 / 52 as "Adjusted Wkly Sal"  
from position;
```

The result looks like this:

3.5 Displaying calculations in columns

SALARY_AMOUNT	CURRENT WKLY SAL	ADJUSTED WKLY SAL
53665.00	1032.019	53684.00
44001.40	846.180	44020.40
45240.00	870.000	45259.00
76440.00	1470.000	76459.00
22048.84	424.016	22067.84
59488.00	1144.000	59507.00
29536.00	568.000	29555.00
<null>	<null>	<null>
<null>	<null>	<null>
50440.50	970.009	50459.50
2200.00	42.307	2219.00
74776.00	1438.000	74795.00
68016.00	1308.000	68035.00
48465.80	932.034	48484.80
36400.00	700.000	36419.00
24000.00	461.538	24019.00
66144.00	1272.000	66163.00
111593.00	2146.019	111612.00
36921.00	710.019	36940.00
48568.48	934.009	48587.48
47009.34	904.025	47028.34
<null>	<null>	<null>
18001.00	346.173	18020.00
42016.00	808.000	42035.00
27976.00	538.000	27995.00
<null>	<null>	<null>
57824.50	1112.009	57843.50
30680.00	590.000	30699.00
76961.00	1480.019	76980.00
19000.10	365.386	19019.10
23920.00	460.000	23939.00
110448.00	2124.000	110467.00
28601.80	550.034	28620.80
<null>	<null>	<null>
85280.00	1640.000	85299.00
<null>	<null>	<null>
59280.00	1140.000	59299.00
<null>	<null>	<null>
<null>	<null>	<null>
<null>	<null>	<null>
47944.00	922.000	47963.00
56977.80	1095.726	56996.80
<null>	<null>	<null>
54184.00	1042.000	54203.00
28912.00	556.000	28931.00
70720.00	1360.000	70739.00
43784.00	842.000	43803.00
117832.68	2266.013	117851.68
146432.00	2816.000	146451.00
28081.40	540.026	28100.40
94953.52	1826.029	94972.52
49921.76	960.033	49940.76
43888.00	844.000	43907.00
22880.00	440.000	22899.00
24441.00	470.019	24460.00
<null>	<null>	<null>
<null>	<null>	<null>
33800.00	650.000	33819.00
24857.00	478.019	24876.00
53665.56	1032.030	53684.56
41600.00	800.000	41619.00
45241.94	870.037	45260.94

61 rows processed

Is the result correct?

Take one salary and do your own pencil and paper calculation to check your answers:

$$24,000.00 + 1,000 = 25,000.00 / 52 = 480.76923$$

The result is wrong: In the calculation involving the increase, the division occurred *before* the addition instead of after. Remember that the default order of evaluation is multiplication and division, performed left to right, and then addition and subtraction, performed left to right.

3.5 Displaying calculations in columns

Use parentheses to specify that you want the addition to take place *before* the division.

Enter:

```
select salary_amount,  
       salary_amount / 52 as "Current Wkly Sal",  
       (salary_amount + 1000) / 52 as "Adjusted Wkly Sal"  
from position;
```

The result looks like this:

SALARY_AMOUNT	CURRENT WKLY SAL	ADJUSTED WKLY SAL
53665.00	1032.019	1051.250
44001.40	846.180	865.411
45240.00	870.000	889.230
76440.00	1470.000	1489.230
22048.84	424.016	443.246
59488.00	1144.000	1163.230
29536.00	568.000	587.230
<null>	<null>	<null>
<null>	<null>	<null>
50440.50	970.009	989.240
2200.00	42.307	61.538
74776.00	1438.000	1457.230
68016.00	1308.000	1327.230
48465.80	932.034	951.265
36400.00	700.000	719.230
24000.00	461.538	480.769
66144.00	1272.000	1291.230
111593.00	2146.019	2165.250
36921.00	710.019	729.250
48568.48	934.009	953.240
47009.34	904.025	923.256
<null>	<null>	<null>
18001.00	346.173	365.403
42016.00	808.000	827.230
27976.00	538.000	557.230
<null>	<null>	<null>
57824.50	1112.009	1131.240
30680.00	590.000	609.230
76961.00	1480.019	1499.250
19000.10	365.386	384.617
23920.00	460.000	479.230
110448.00	2124.000	2143.230
28601.80	550.034	569.265
<null>	<null>	<null>
85280.00	1640.000	1659.230
<null>	<null>	<null>
59280.00	1140.000	1159.230
<null>	<null>	<null>
<null>	<null>	<null>
47944.00	922.000	941.230
56977.80	1095.726	1114.957
<null>	<null>	<null>
54184.00	1042.000	1061.230
28912.00	556.000	575.230
70720.00	1360.000	1379.230
43784.00	842.000	861.230
117832.68	2266.013	2285.243
146432.00	2816.000	2835.230
28081.40	540.026	559.257
94953.52	1826.029	1845.260
49921.76	960.033	979.264
43888.00	844.000	863.230
22880.00	440.000	459.230
24441.00	470.019	489.250
<null>	<null>	<null>
<null>	<null>	<null>
33800.00	650.000	669.230
24857.00	478.019	497.250
53665.56	1032.030	1051.260
41600.00	800.000	819.230
45241.94	870.037	889.268

61 rows processed

How does this result match your written calculation?

If you did your written calculation correctly, it should match this result.

3.6 Eliminating duplicate rows

Sometimes a row in a selected column contains information that is the same as information in another row.

Why duplicates occur: The EXPERTISE table contains skill IDs, the IDs of employees who have the skills, the level of ability an employee has in a skill, and the date the ability was acquired. Commonwealth Auto may have several employees who match a particular skill in the SKILL table and may have no employees who match another skill.

How it's done: To obtain a list of skill IDs *associated with at least one employee*, enter:

```
select skill_id
       from expertise;
```

This gives a list of skill IDs that have been matched to employees who have that skill. Any skill that has no employees associated with it will not show up in the result.

The result looks like this:

3.6 Eliminating duplicate rows

```
SKILL_ID
-----
1000
6470
1000
6770
6770
7000
3065
3333
6770
4430
7000
5309
1000
6670
6470
3333
4444
7000
4250
4370
5180
1030
4490
5200
6666
5420
5430
1000
5500
5309
5180
1000
4430
3333
6650
6670
6770
6770
5309
5500
6650
5200
7000
7000
7000
5309
5200
6666
4370
4410
7000
7000
4370
4410
4420
7000
1000
4430
5500
3065
```

```
6670
7000
4250
5130
5309
5130
6770
7000
5200

69 rows processed
```

Why are some of the skill IDs repeated?

The result shows the skill ID for each employee. If more than one employee has that particular skill, the skill ID is repeated.

3.6 Eliminating duplicate rows

To see this more clearly, look at the skills and the associated employees by entering:

```
select skill_id, emp_id
       from expertise;
```

The result looks like this:

SKILL_ID	EMP_ID
1000	1003
6470	1034
1000	1234
6770	1765
6770	2004
7000	2010
3065	2096
3333	2096
6770	2106
4430	2174
7000	2180
5309	2209
1000	2246
6670	2246
6470	2424
3333	2437
4444	2437
7000	2448
4250	2461
4370	2461
5180	2461
1030	2466
4490	2466
5200	2466
6666	2598
5420	2781
5430	2781
1000	2894
5500	2894
5309	3082
5180	3118
1000	3222
4430	3222
3333	3288

```
6650 3288
6670 3288
6770 3294
6770 3338
5309 3341
5500 3411
6650 3433
5200 3449
7000 3704
7000 3764
7000 3767
5309 3769
5200 3778
6666 3778
4370 3841
4410 3841
7000 3991
7000 4001
4370 4002
4410 4002
4420 4008
7000 4027
1000 4321
4430 4321
5500 4358
3065 4456
6670 4456
7000 4660
4250 4703
5130 4703
5309 4773
5130 4962
6770 5008
7000 5090
5200 5103
```

69 rows processed

You want to eliminate the duplicate rows resulting from your first `SELECT` statement in order to see each skill ID only once.

To eliminate these rows, you can use the `DISTINCT` option immediately after the word `SELECT`.

How it's done with `DISTINCT`: Using the first `SELECT` statement, add `DISTINCT` after `SELECT`:

```
select distinct skill_id
       from expertise;
```

The result looks like this:

```
SKILL_ID
-----
1000
1030
3065
3333
4250
4370
4410
4420
4430
4444
4490
5130
5180
5200
5309
5420
5430
5500
6470
6650
6666
6670
6770
7000

24 rows processed
```

Now the result shows a list of skill IDs with no duplicates.

Since using `DISTINCT` eliminates duplicate rows, fewer rows are returned.

3.6.1 Exercise 3-6

Now you try it: The Accounting department needs a list of communities represented by employees in this company in order to identify applicable city taxes. One of the columns in the `EMPLOYEE` table contains the name of the community in which the employee resides. Enter a `SELECT` statement to list the communities in the table without showing any duplicates.

The result looks like this:

```
CITY
----
Boston
Brookline
Camden
Canton
Concord
Dedham
Grover
Medford
Natick
Newton
Stoneham
Taunton
Wilmington

13 rows processed
```

If your results do not match what you see above, check “Exercise 3-6 answer” on page B-5 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

3.7 Organizing data

When you retrieve data from the database, the rows are in an order selected by the database management system. If you want the data sorted in a particular order, use the `ORDER BY` clause. The `ORDER BY` clause must be the last clause in a `SELECT` statement.

How it's done: You can order an employee list by entering:

```
select emp_id, emp_lname
       from employee
       order by emp_lname;
```

The result looks like this:

```
EMP_ID  EMP_LNAME
-----  -
2180    Albertini
1765    Alexander
2461    Anderson
1003    Baldwin
2466    Bennett
4321    Bradley
3082    Brooks
2096    Carlson
2145    Catlin
4008    Clark
4027    Courtney
3433    Crane
3841    Cromwell
4773    Dexter
3769    Donelson
5103    Ferguson
3778    Ferndale
5008    Fordman
1034    Gallway
2894    Griffin
4703    Halloran
2246    Hamel
2598    Jacobs
2004    Johnson
3294    Johnson
3199    Loren
3767    Lowe
2448    Lynn
4660    MacGregor
1234    Mills
3704    Moore
3764    Park
2010    Parker
4358    Robinson
4002    Roy
3288    Sampson
2209    Smith
3341    Smith
2299    Spade
3449    Taylor
4001    Thompson
2437    Thompson
4456    Thompson
2781    Thurston
2898    Umidy
3222    Voltmer
3338    White
4962    White
2106    Widman
2424    Wilder
3991    Wilkins
3411    Williams
5090    Wills
3118    Wooding
2174    Zander
```

```
55 rows processed
```

You can specify either ascending (ASC) or descending (DESC) order. The default order is ascending.

If you sort on a column that contains null values, the null values are grouped together.

The column you choose to order by must also be in the column list after SELECT.

3.7.1 Exercise 3-7

Now you try it: Change the example above so that it is sorted in descending order. Specify DESC after the column name in the ORDER BY clause.

The result looks like this:

```
EMP_ID  EMP_LNAME
-----  -
2174    Zander
3118    Wooding
5090    Wills
3411    Williams
3991    Wilkins
2424    Wilder
2106    Widman
3338    White
4962    White
3222    Voltmer
2898    Umidy
2781    Thurston
2437    Thompson
4456    Thompson
4001    Thompson
3449    Taylor
2299    Spade
2209    Smith
3341    Smith
3288    Sampson
4002    Roy
4358    Robinson
2010    Parker
3764    Park
3704    Moore
1234    Mills
4660    MacGregor
2448    Lynn
3767    Lowe
3199    Loren
2004    Johnson
3294    Johnson
2598    Jacobs
2246    Hamel
4703    Halloran
2894    Griffin
1034    Gallway
5008    Fordman
3778    Ferndale
5103    Ferguson
3769    Doneison
4773    Dexter
3841    Cromwell
3433    Crane
4027    Courtney
4008    Clark
2145    Catlin
2096    Carlson
3082    Brooks
4321    Bradley
2466    Bennett
1003    Baldwin
2461    Anderson
1765    Alexander
2180    Albertini

55 rows processed
```

If your results do not match what you see above, check “Exercise 3-7 answer” on page B-5 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

3.7.2 Exercise 3-8

Try another: Display the skills available in the company, as you did earlier, but list them in numeric order to make it easier to scan. Show both skill ID and skill name from the SKILL table.

The result looks like this:

```

SKILL_ID SKILL_NAME
-----
1000 Management
1030 Acct Mgt
3065 Electronics
3088 Brake work
3333 Bodywork
4250 Data Entry
4370 Filing
4410 Typing
4420 Telephone
4430 Interviewing
4444 Assembly
4490 Gen Ledger
5130 Basic Math
5160 Calculus
5180 Statistics
5200 Gen Acctng
5309 Appraising
5420 Writing
5430 Mktng Writing
5500 Gen Mktng
6470 Window Installation
6650 Diesel Engine Repair
6666 Billing
6670 Gas Engine Repair
6770 Purchasing
7000 Sales

```

26 rows processed

If your results do not match what you see above, check “Exercise 3-8 answer” on page B-5 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

Multiple sort columns: You've seen how to sort selected rows by specifying ORDER BY and a column name. If you specify more than one column name after ORDER BY, SQL sorts rows by the first column named, then by the second column named, and so on.

How it's done: You often want to sort all employees by first name within last name for an employee list. To do this, enter:

```

select emp_lname, emp_fname
       from employee
       order by emp_lname, emp_fname;

```

The result looks like this:

EMP_LNAME	EMP_FNAME
Albertini	Joan
Alexander	David
Anderson	Alice
Baldwin	James
Bennett	Patricia
Bradley	George
Brooks	John
Carlson	Thomas
Catlin	Martin
Clark	Robert
Courtney	Cecile
Crane	Herbert
Cromwell	Michelle
Dexter	Janice
Donelson	Julie
Ferguson	Adele
Ferndale	Jane
Fordman	Timothy
Gallway	James
Griffin	William
Halloran	Martin
Hamel	Marylou
Jacobs	Mary
Johnson	Carolyn
Johnson	Eleanor
Loren	Martin
Lowe	Frank
Lynn	David
MacGregor	Bruce
Mills	Thomas
Moore	Richard
Park	Deborah
Parker	Cora
Robinson	Judith
Roy	Linda
Sampson	Ralph
Smith	Carl
Smith	Michael
Spade	Samuel
Taylor	Cynthia
Thompson	Henry
Thompson	Jason
Thompson	Thomas
Thurston	Joseph
Umidy	Mary
Voltmer	Louise
White	Mark
White	Peter
Widman	Susan
Wilder	Ronald
Wilkins	Fred
Williams	Catherine
Wills	Stephen
Wooding	Alan
Zander	Jonathan

55 rows processed

3.7.3 Exercise 3-9

Now you try it: Management needs a list of all employees assigned to each department. Enter a `SELECT` statement to show the department ID, last name, and employee ID from the `EMPLOYEE` table. Order the list by last name within each department. Use the table descriptions in Appendix C, “Table Descriptions” to find the correct column names.

The result looks like this:

DEPT_ID	EMP_LNAME	EMP_ID
1100	Fordman	5008
1100	Hamel	2246
1100	Halloran	4703
1110	Widman	2106
1110	Alexander	1765
1120	White	3338
1120	Johnson	3294
1120	Johnson	2004
1120	Umidy	2898
2200	Moore	3704
2200	Lowe	3767
2200	Albertini	2180
2200	Lynn	2448
2200	MacGregor	4660
2210	Wills	5090
2210	White	4962
2210	Park	3764
2210	Thompson	4001
2210	Courtney	4027
2210	Clark	4008
2210	Parker	2010
2210	Wilkins	3991
3510	Dexter	4773
3510	Brooks	3082
3520	Doneison	3769
3530	Smith	3341
3530	Smith	2209
4500	Zander	2174
4500	Voltmer	3222
4500	Wooding	3118
4600	Thompson	2437
4600	Gallway	1034
4600	Crane	3433
4600	Thompson	4456
4600	Carlson	2096
4600	Spade	2299
4600	Loren	3199
4600	Wilder	2424
4600	Sampson	3288
5000	Ferguson	5103
5000	Taylor	3449
5000	Bennett	2466
5100	Ferndale	3778
5100	Jacobs	2598
5200	Griffin	2894
5200	Catlin	2145
5200	Thurston	2781
5200	Williams	3411
5200	Robinson	4358
6200	Cromwell	3841
6200	Mills	1234
6200	Anderson	2461
6200	Bradley	4321
6200	Roy	4002
6200	Baldwin	1003

55 rows processed

If your results do not match what you see above, check “Exercise 3-9 answer” on page B-5 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

Identifying columns by number: Each column named after SELECT has an assumed number corresponding to the order in which it is named. You can use this number to identify columns in the ORDER BY clause:

```
select emp_id, emp_fname, emp_lname from employee
      ↑       ↑       ↑
      1       2       3
      order by 3;
```

How it's done: Earlier you wrote a SELECT statement to display the skills available in the company in numeric order by skill ID. Modify the ORDER BY clause in that statement to identify the column by number rather than by name:

```
select skill_id, skill_name
       from skill
       order by 1;
```

The result looks like this:

```
SKILL_ID SKILL_NAME
-----
1000 Management
1030 Acct Mgt
3065 Electronics
3088 Brake work
3333 Bodywork
4250 Data Entry
4370 Filing
4410 Typing
4420 Telephone
4430 Interviewing
4444 Assembly
4490 Gen Ledger
5130 Basic Math
5160 Calculus
5180 Statistics
5200 Gen Acctng
5309 Appraising
5420 Writing
5430 Mktng Writing
5500 Gen Mktng
6470 Window Installation
6650 Diesel Engine Repair
6666 Billing
6670 Gas Engine Repair
6770 Purchasing
7000 Sales

26 rows processed
```

You *must* refer to a column by number when the column is a calculated column like **salary_amount / 52**.

You can also use a column number when you select all columns from a table with SELECT *.

For each of these, the result looks like this:

```
CONSULTANT  HOURLY RATE
-----
      9000      148.00
      9388       76.00
      9443       50.00
      9439       47.00

4 rows processed
```

You can use any of these methods or a combination of them in an ORDER BY clause.

3.8 Review

Choose the correct answers for the questions below. More than one answer can apply for each question.

1. How many columns would be retrieved by the following statement:

```
select * from sample_table;
```

 - a. One
 - b. Five
 - c. All
2. How can you limit the number of columns returned by your SELECT statement?
 - a. Use a FROM clause
 - b. List the columns you want to see
 - c. Use * after SELECT
 - d. Leave out the * between SELECT and FROM
3. How can you give heading names to columns?
 - a. Put a comma after the column name and specify the heading
 - b. Use AS after the column name and specify the heading
 - c. Use the heading in place of the column name
4. Given a table called SUPPLY_PRICE and a column in that table called PART_NUMBER, which of the following statements will find the number of *unique* part numbers in the table?
 - a. **select part_number from supply_price;**
 - b. **select * from supply_price;**
 - c. **select distinct part_number from supply_price;**
 - d. **select part_number distinct from supply_price;**
5. How can you name the column you want to sort by?
 - a. Use the column name
 - b. Use the heading name
 - c. Use the table name
 - d. Use the column number

To check your answers, see “Review answers” on page B-6.

3.9 Scenarios

Create the appropriate statements online to retrieve the needed data:

1. You need to list all jobs the company has for a government report. The report should show job ID, job title, and minimum and maximum rate for the job. Use the JOB table, checking Appendix C, “Table Descriptions” for table descriptions.
2. The report you just created (in Scenario 1) has all the necessary information but is difficult to read because it is not sorted. Modify the SELECT statement to create the same report sorted by job title.
3. Periodically, a company list is produced showing each department and all employees assigned to that department. This list should be sorted first by department ID and then by employee ID within each department. Display department ID, employee ID, and employee last name. Create the appropriate SQL SELECT statement to produce this list using the EMPLOYEE table.
4. The report you just created is very useful except that the headings are difficult to understand. Rewrite the statement so that the column headings are "Department", "Employee ID", and "Last Name".
5. The Human Resources department needs a report of all employees listing ID, amount of vacation accrued for each employee, and vacation accrued incremented by 32 hours, in order to see whether any employees will have over the maximum allowable vacation once the accruals have been applied. Create this report using the BENEFITS table. Name the column headings appropriately, and order the report by employee ID.

To check your answers, see “Scenario answers” on page B-6.

Chapter 4. Using Conditional Retrieval

4.1 About this chapter	4-3
4.2 The WHERE clause	4-4
4.3 Comparison operators and keywords in predicates	4-5
4.4 Using comparison operators in predicates	4-6
4.4.1 Exercise 4-1	4-7
4.4.2 Exercise 4-2	4-8
4.4.3 Exercise 4-3	4-9
4.4.4 Exercise 4-4	4-9
4.5 Using keywords in predicates	4-11
4.5.1 Exercise 4-5	4-11
4.5.2 Exercise 4-6	4-13
4.5.3 Exercise 4-7	4-15
4.5.4 Exercise 4-8	4-17
4.5.5 Exercise 4-9	4-19
4.5.6 Exercise 4-10	4-19
4.6 Using calculated values in predicates	4-20
4.6.1 Exercise 4-11	4-21
4.7 Combining predicates	4-22
4.7.1 Exercise 4-12	4-22
4.8 Review	4-27
4.9 Scenarios	4-29

4.1 About this chapter

Goal: When you have completed this chapter, you will be able to create SQL statements to retrieve selected rows of data from a table and to use multiple predicates in a WHERE clause. You will also understand the order in which multiple conditions are evaluated.

Summary: Usually you want to retrieve only some rows from a table just as you want to retrieve only some columns in that table. You have already seen how to limit the number of **columns** displayed. Now you will see how to limit the number of **rows** displayed using a WHERE clause. A WHERE clause specifies criteria used in selecting rows to be retrieved.

4.2 The WHERE clause

You can screen the data being retrieved by using the **WHERE** clause in a **SELECT** statement. In the **WHERE** clause, you specify selection criteria. The **WHERE** clause filters out rows that do not meet the selection criteria. The **WHERE** clause follows the **FROM** clause:

```
select ...  
    from ...  
    where ...  
    order by ...;
```

Components of the WHERE clause: The **WHERE** clause is made up of two components:

- The keyword **WHERE**
- A search condition

A search condition is made up of predicates. In general, predicates compare values to one another. If the values meet the comparison, the row is selected.

4.3 Comparison operators and keywords in predicates

You use a predicate to compare one value to another. The values compared in the predicate must be of compatible data types. For example, a column defined as a character data type cannot be compared to a column defined as an integer data type even though the character string contains numbers.

Check the data type when comparing: The table layout specifies whether the column you want to compare is defined as a character data type. A column may appear to contain numeric data when you look at the values stored in the database, although it is actually defined as character. For example, the `SKILL_LEVEL` column in the `EXPERTISE` table is defined as a character column but usually contains numeric data.

Entering character literals: Use single quotation marks if you use a character literal such as 'Smith' and enter the literal with uppercase and lowercase letters exactly as you expect it exists in the database.

Ways you can compare values: You can use these operators to compare values:

Comparison operator	Meaning
=	Equal to
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

You can use these keywords to compare values:

Keyword	Meaning
IS NULL	Checks whether a value is null
BETWEEN	Locates values within a range of values
IN	Locates values identified by specific values in a list
LIKE	Retrieves rows based on character combinations in a nonnumeric column

4.4 Using comparison operators in predicates

The search value in the predicate can be:

- A numeric literal (for example, `vac_accrued >= 40`).
- A character literal (for example, `emp_lname = 'Smith'`).

You can also use two column names in a predicate (for example, `vac_taken <= vac_accrued`).

The column name you use in the WHERE clause need not appear in the SELECT column list.

How it's done: The Human Resources department needs to see a list of all employees and the number of vacation hours they have accrued in 1999. The BENEFITS table contains this information. Enter:

```
select emp_id, vac_accrued
       from benefits
       where fiscal_year = 2000;
```

The result looks like this:

EMP_ID	VAC_ACCRUED
3411	68.00
4773	68.00
2010	92.75
3338	68.00
2246	92.50
1034	92.50
2424	92.50
3767	68.00
3449	68.00
3082	68.00
3341	68.00
4660	68.00
2209	92.50
2894	68.00
4001	68.00
5090	46.00
1765	92.50
4456	68.00
3991	68.00
3778	68.00
4358	68.00
4962	68.00
2180	92.50
2106	92.50
3222	68.00
4002	68.00
2437	68.00
2096	92.50
2004	92.50
5103	46.00
5008	46.50
4321	68.00
2598	60.00
3764	68.00
2448	68.00
2461	68.00
1003	92.00
1234	92.00
4027	68.00
2466	92.50
2174	92.00
2781	68.00
3704	68.00
4008	68.00
3433	68.00
3288	68.00
3841	68.00
4703	46.75
3294	68.00
3769	68.00
3118	68.00

51 rows processed

4.4.1 Exercise 4-1

Now you try it: The manager of the Marketing department is looking at a resource plan that assigns employee 5103 to a project coming up. However, the manager doesn't know who employee 5103 is.

Create a SELECT statement to retrieve all rows in the EMPLOYEE table that have an employee with an ID of 5103. Display employee ID and first and last name.

The result looks like this:

EMP_ID	EMP_FNAME	EMP_LNAME
5103	Adele	Ferguson

1 row processed

If your results do not match what you see above, check “Exercise 4-1 answer” on page B-10 for the correct SQL syntax.

Why do you see only one row?

The employee ID is a unique key, so there is only one employee with this ID.

4.4.2 Exercise 4-2

Try another: The new company fiscal year is approaching and the Accounting department's budget director is compiling some salary statistics.

The budget director needs a list of employee IDs, job IDs, and salaries where the salary is greater than \$100,000. The POSITION table contains this information.

The result looks like this:

EMP_ID	JOB_ID	SALARY_AMOUNT
2894	6021	111593.00
3222	6004	110448.00
1234	8001	117832.68
1003	9001	146432.00

4 rows processed

If your results do not match what you see above, check “Exercise 4-2 answer” on page B-10 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

4.4.3 Exercise 4-3

And another: Commonwealth Auto has employees who live in several different communities. The Accounting department needs to know which employees live in Boston because a city tax is being levied on those residents and the payroll must be adjusted.

Use the EMPLOYEE table to write a SELECT statement that selects all employees who live in Boston. Type in the value exactly as you expect to find it in the table, using uppercase and lowercase letters. Enclose the value in quotes because CITY is defined as a character column. Display employee ID, first and last name, and city and order the result by employee ID.

The result looks like this:

EMP_ID	EMP_FNAME	EMP_LNAME	CITY
1003	James	Baldwin	Boston
2010	Cora	Parker	Boston
2437	Henry	Thompson	Boston
3411	Catherine	Williams	Boston
3841	Michelle	Cromwell	Boston
4962	Peter	White	Boston

6 rows processed

If your results do not match what you see above, check “Exercise 4-3 answer” on page B-10 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

4.4.4 Exercise 4-4

Try it using NOT: Use NOT following the word WHERE to select rows that do not meet the search condition.

Employees who do not live in Boston are not affected by the new tax. Create another SELECT statement to retrieve all employees who do not live in Boston. Use the keyword NOT after WHERE. Display employee ID, first and last name, and city.

The result looks like this:

EMP_ID	EMP_FNAME	EMP_LNAME	CITY
2299	Samuel	Spade	Canton
4773	Janice	Dexter	Medford
3338	Mark	White	Canton
2246	Marylou	Hamel	Medford
1034	James	Galloway	Stoneham
2424	Ronald	Wilder	Natick
3767	Frank	Lowe	Natick
2898	Mary	Umidy	Medford
3449	Cynthia	Taylor	Concord
3082	John	Brooks	Camden
3341	Carl	Smith	Newton
3199	Martin	Loren	Grover
4660	Bruce	MacGregor	Camden
2209	Michael	Smith	Brookline
2894	William	Griffin	Taunton
4001	Jason	Thompson	Natick
5090	Stephen	Wills	Canton
1765	David	Alexander	Grover
4456	Thomas	Thompson	Newton
2145	Martin	Catlin	Wilmington
3991	Fred	Wilkins	Taunton
3778	Jane	Ferndale	Medford
4358	Judith	Robinson	Wilmington
2180	Joan	Albertini	Medford
2106	Susan	Widman	Medford
3222	Louise	Voltmer	Brookline
4002	Linda	Roy	Wilmington
2096	Thomas	Carlson	Brookline
2004	Eleanor	Johnson	Medford
5103	Adele	Ferguson	Brookline
5008	Timothy	Fordman	Brookline
4321	George	Bradley	Grover
2598	Mary	Jacobs	Camden
3764	Deborah	Park	Brookline
2461	Alice	Anderson	Medford
2448	David	Lynn	Natick
1234	Thomas	Mills	Brookline
2466	Patricia	Bennett	Medford
4027	Cecile	Courtney	Natick
2174	Jonathan	Zander	Brookline
2781	Joseph	Thurston	Stoneham
3704	Richard	Moore	Dedham
4008	Robert	Clark	Brookline
3433	Herbert	Crane	Newton
3288	Ralph	Sampson	Newton
4703	Martin	Halloran	Brookline
3294	Carolyn	Johnson	Brookline
3118	Alan	Wooding	Canton
3769	Julie	Doneison	Grover

49 rows processed

If your results do not match what you see above, check “Exercise 4-4 answer” on page B-10 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

What is another way of getting this information?

You could have used \neq (not equal) instead of the keyword NOT.

4.5 Using keywords in predicates

You have been using comparison operators in predicates to compare one value to another. You can also use keywords in predicates.

Testing for null values: You use the keyword IS NULL to retrieve rows for which no value has been stored in the specified column.

How it's done: The company needs to have a list of employees who do not have a home telephone so that the Human Resources department can get in touch with them by other means in case of emergency.

To retrieve all employees who do not have a telephone, enter:

```
select emp_id, phone
       from employee
       where phone is null;
```

The result looks like this:

```
EMP_ID  PHONE
-----  -----
      2299 <null>
      3411 <null>
      2010 <null>
      3199 <null>
      2598 <null>
      4008 <null>

6 rows processed
```

4.5.1 Exercise 4-5

Now you try it: Only sales employees at Commonwealth Auto receive bonuses as part of their earnings. Payroll wants a list of all employees who do not receive bonuses.

Enter a SELECT statement to retrieve all employees from the POSITION table for which the database does not have a bonus percentage. Display employee ID.

The result looks like this:

```
EMP_ID
-----
3411
3411
4773
3338
2246
2246
1034
2424
3767
3449
3082
3341
4660
2209
2894
1765
4456
1765
3991
3778
4358
4962
2180
2180
2106
3222
4002
2437
2096
2096
2004
2004
5103
5008
4321
2598
3764
2461
1234
1003
4027
2466
2174
2781
3704
4008
3433
3288
3841
4703
3294
3769
3118

53 rows processed
```

If your results do not match what you see above, check “Exercise 4-5 answer” on page B-10 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

IS NOT NULL: Use IS NOT NULL to *eliminate* rows containing nulls in a specified column.

4.5.2 Exercise 4-6

Now you try it: Commonwealth Auto needs to be able to reach employees in case of emergency. The Human Resources department knows that not all employees have telephones. They need to have a list of telephone numbers for those employees who *do* have telephones.

Enter the SELECT statement to list all employees from the EMPLOYEE table who have a telephone. Display employee ID, first and last name, and telephone.

The result looks like this:

EMP_ID	EMP_FNAME	EMP_LNAME	PHONE
-----	-----	-----	-----
4773	Janice	Dexter	5083847566
3338	Mark	White	6179238844
2246	Marylou	Hamel	5083457789
1034	James	Gallway	6172251178
2424	Ronald	Wilder	5083347700
3767	Frank	Lowe	5082844094
2898	Mary	Umidy	6173458860
3449	Cynthia	Taylor	5082684508
3082	John	Brooks	5089273644
3341	Carl	Smith	6179658099
4660	Bruce	MacGregor	5092344620
2209	Michael	Smith	6175563331
2894	William	Griffin	5088449008
4001	Jason	Thompson	5082649956
1765	David	Alexander	5087394772
4456	Thomas	Thompson	6179660089
2145	Martin	Catlin	5087486625
3991	Fred	Wilkins	5081840883
3778	Jane	Ferndale	6173450099
4358	Judith	Robinson	5087488011
4962	Peter	White	6177732280
2180	Joan	Albertini	5083145366
2106	Susan	Widman	5083346364
3222	Louise	Voltmer	6176635520
4002	Linda	Roy	5088477701
2437	Henry	Thompson	6179264105
2096	Thomas	Carlson	6175553643
2004	Eleanor	Johnson	5089253998
5103	Adele	Ferguson	6176600684
5008	Timothy	Fordman	6176642209
4321	George	Bradley	5087463300
3764	Deborah	Park	6179458377
2461	Alice	Anderson	5083873664
2448	David	Lynn	5082844736
1003	James	Baldwin	6173295757
1234	Thomas	Mills	6176646602
2466	Patricia	Bennett	5089487709
4027	Cecile	Courtney	5089445386
2174	Jonathan	Zander	6176633854
2781	Joseph	Thurston	6173286008
3704	Richard	Moore	6177739440
3841	Michelle	Cromwell	6173298763
3433	Herbert	Crane	6178653440
3288	Ralph	Sampson	6179654443
4703	Martin	Halloran	6176648290
3294	Carolyn	Johnson	6175567551
3118	Alan	Wooding	5083766984
3769	Julie	Donelson	5084850432
49 rows	processed		

If your results do not match what you see above, check “Exercise 4-6 answer” on page B-10 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

BETWEEN predicate: Use the BETWEEN predicate to specify a range of values you are searching for. BETWEEN selects all rows that have values in a specified column in between or equal to the starting or ending values of the specified range.

How it's done: Human Resources is interested in obtaining a list of employees who have between 1 and 3 dependents being covered by their insurance plan.

To retrieve this data, enter:

```
select emp_id, num_dependents
       from coverage
       where num_dependents between 1 and 3;
```

The result looks like this:

EMP_ID	NUM_DEPENDENTS
2299	1
3411	3
3411	3
3338	2
2246	2
2246	2
3767	2
3767	2
3199	2
3199	2
2894	3
2894	3
5090	3
4456	1
1765	2
1765	2
4358	1
4358	1
3222	2
2437	2
2096	1
2096	3
2096	3
5103	1
5103	1
5008	2
5008	2
2598	1
2448	3
1003	3
1003	3
2781	2
4008	1
3704	3
3433	1
3433	1
3433	1
3288	1
3288	1
4703	1
4703	1
3118	1

42 rows processed

4.5.3 Exercise 4-7

Now you try it: Human Resources is doing a salary comparison and needs some information on jobs, employees, and salaries. They have asked you to show them all employees whose salary is between \$20,000 and \$35,000.

What table holds this information?

The information is stored in the POSITION table.

Enter the appropriate SELECT statement.

The result looks like this:

JOB_ID	EMP_ID	SALARY_AMOUNT
2077	3338	22048.84
2077	2246	29536.00
3333	4660	24000.00
3333	3991	27976.00
3333	4962	30680.00
2077	2106	23920.00
4012	4002	28601.80
3333	3764	28912.00
3333	4027	28081.40
3333	3704	22880.00
3333	4008	24441.00
4012	3841	33800.00
2077	4703	24857.00

13 rows processed

If your results do not match what you see above, check “Exercise 4-7 answer” on page B-10 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

Use NOT BETWEEN to retrieve all rows that do *not* fall into the specified range.

IN predicate: You can compare a value to a list of values using the IN predicate.

How it's done: Human Resources needs to identify employees who reside in three communities for potential car pooling. To do this, enter:

```
select emp_id, city
       from employee
       where city in ('Camden', 'Brookline', 'Canton');
```

Enclose the list of values in parentheses and separate values with a comma.

The result looks like this:

```

EMP_ID  CITY
-----  ----
 2299  Canton
 3338  Canton
 3082  Camden
 4660  Camden
 2209  Brookline
 5090  Canton
 3222  Brookline
 2096  Brookline
 5103  Brookline
 5008  Brookline
 2598  Camden
 3764  Brookline
 1234  Brookline
 2174  Brookline
 4008  Brookline
 4703  Brookline
 3294  Brookline
 3118  Canton

18 rows processed

```

4.5.4 Exercise 4-8

Now you try it: The Payroll department needs to identify employees whose salaries are \$41,600, \$45,240, or \$50,440 for tax rate purposes. Using the BENEFITS table, write a SELECT statement to display employee ID and salary.

The result looks like this:

```

EMP_ID  SALARY_AMOUNT
-----  -----
 4773   45240.00
 3769   41600.00

2 rows processed

```

If your results do not match what you see above, check “Exercise 4-8 answer” on page B-10 for the correct SQL syntax.

You can insert NOT before IN to identify values you do *not* want returned.

The LIKE predicate and masks: You can use the keyword LIKE and mask characters to find a character string when you know or are concerned about only some of the characters. Mask characters are symbols that serve as place holders for other characters. For example, **LIKE 'Th%'** means everything beginning with **Th**. The % is the mask specifying that any number of characters can follow.

This table shows the symbols you can use as mask characters:

Mask	Meaning
Percent (%)	Specifies any number of unknown characters (including none)
Underscore (_)	Specifies a single unknown character

How it's done: The company nurse wants to identify all employees whose last names begin with **S** in order to notify them that their yearly physical examination is due. Enter:

```
select emp_lname
       from employee
       where emp_lname like 'S%';
```

Enter the character string to be matched exactly as you expect to find it in the database. Use uppercase and lowercase letters as necessary.

The result looks like this:

```
EMP_LNAME
-----
Sampson
Smith
Smith
Spade

4 rows processed
```

Where to place mask characters: Mask characters can come anywhere in the search string.

If you want to find all employees whose names contain **mp**, enter:

```
select emp_lname
       from employee
       where emp_lname like '%mp%';
```

The result looks like this:

```
EMP_LNAME
-----
Sampson
Thompson
Thompson
Thompson

4 rows processed
```

4.5.5 Exercise 4-9

Now you try it: How would you display all departments associated with **NEW CARS**?

The result looks like this:

```

DEPT_ID  DEPT_NAME
-----  -
 4200    LEASING - NEW CARS
 2210    SALES - NEW CARS
 3520    APPRAISAL NEW CARS
 1110    PURCHASING - NEW CARS

4 rows processed

```

If your results do not match what you see above, check “Exercise 4-9 answer” on page B-10 for the correct SQL syntax. Remember, result tables may be shortened in this guide.

Use NOT in front of LIKE to search for rows containing all values *except* those that match the mask.

4.5.6 Exercise 4-10

Try another: Enter a SELECT statement to display all departments that are *not* associated with associated with **NEW CARS**.

The result looks like this:

```

DEPT_ID  DEPT_NAME
-----  -
 1120    PURCHASING - SERVICE
 4900    MIS
 5000    CORPORATE ACCOUNTING
 4500    HUMAN RESOURCES
 4600    MAINTENANCE
 2200    SALES - USED CARS
 5100    BILLING
 6200    CORPORATE ADMINISTRATION
 3530    APPRAISAL - SERVICE
 6000    LEGAL
 3510    APPRAISAL - USED CARS
 1100    PURCHASING - USED CARS
 5200    CORPORATE MARKETING

13 rows processed

```

If your results do not match what you see above, check “Exercise 4-10 answer” on page B-10 for the correct SQL syntax. Remember, result tables may be shortened in this guide.

4.6 Using calculated values in predicates

You can use arithmetic expressions to calculate a value for a search condition. Use the following symbols for arithmetic operations:

Symbol	Meaning
*	Multiplication
/	Division
+	Addition
-	Subtraction

Remember that multiplication and division are performed first, from left to right, and addition and subtraction second, from left to right. You can control the order in which operations are performed by using parentheses to enclose the operations you want performed first.

How it's done: As part of the salary review process, Human Resources needs to identify all jobs where the difference between the maximum and minimum salaries is greater than \$10,000. To do this, use the JOB table and enter:

```
select job_id, job_title, min_rate,
       max_rate, max_rate - min_rate
from job
where max_rate - min_rate > 10000;
```

The result looks like this:

JOB_ID	JOB_TITLE	MIN_RATE	MAX_RATE	(EXPR)
8001	Vice President	90000.00	136000.00	46000.00
2077	Purch Clerk	17000.00	30000.00	13000.00
9001	President	111000.00	190000.00	79000.00
4700	Purch Agnt	33000.00	60000.00	27000.00
3029	Computer Operator	25000.00	44000.00	19000.00
6011	Manager - Acctng	59400.00	121000.00	61600.00
4130	Benefits Analyst	35000.00	56000.00	21000.00
4666	Sr Mechanic	41000.00	91000.00	50000.00
4123	Recruiter	35000.00	56000.00	21000.00
5555	Salesperson	30000.00	79000.00	49000.00
4025	Writer - Mktng	31000.00	50000.00	19000.00
4023	Accountant	44000.00	120000.00	76000.00
4734	Mktng Admin	25000.00	62000.00	37000.00
5110	CUST SER MGR	40000.00	108000.00	68000.00
6004	Manager - HR	66000.00	138000.00	72000.00
5111	CUST SER REP	27000.00	54000.00	27000.00
4012	Admin Asst	21000.00	44000.00	23000.00
2055	PAYROLL CLERK	17000.00	30000.00	13000.00
5890	Appraisal Spec	45000.00	70000.00	25000.00
3333	Sales Trainee	21600.00	39000.00	17400.00
6021	Manager - Mktng	76000.00	150000.00	74000.00

21 rows processed

4.6.1 Exercise 4-11

Now you try it: The corporate planning group wants to know what projects have been completed in less time than originally estimated.

Use the PROJECT table to identify the columns and display project ID as well as the number of hours saved.

The result looks like this:

PROJ_ID	(EXPR)
C203	58.50

1 row processed

If your results do not match what you see above, check “Exercise 4-11 answer” on page B-11 for the correct SQL syntax.

4.7 Combining predicates

You can combine predicates with AND and OR:

- Use AND when all predicates must be true
- Use OR when only one predicate must be true

The default order of evaluation is AND before OR. You can use parentheses to override the default order.

How it's done: The Human Resources department needs to identify employees who live in Camden, Brookline, or Canton and who have telephones in order to set up a calling network. You created both of these search conditions earlier in this chapter. Now you want to combine them.

To produce this list, enter:

```
select emp_id, city, phone
       from employee
       where city in ('Camden', 'Brookline', 'Canton')
              and phone is not null;
```

The result looks like this:

EMP_ID	CITY	PHONE
3338	Canton	6179238844
3082	Camden	5089273644
4660	Camden	5092344620
2209	Brookline	6175563331
5090	Canton	5083389935
3222	Brookline	6176635520
2096	Brookline	6175553643
5103	Brookline	6176600684
5008	Brookline	6176642209
3764	Brookline	6179458377
1234	Brookline	6176646602
2174	Brookline	6176633854
4703	Brookline	6176648290
3294	Brookline	6175567551
3118	Canton	5083766984

15 rows processed

4.7.1 Exercise 4-12

Now you try it: Change this SELECT statement from AND to OR.

Your result looks like this:

EMP_ID	CITY	PHONE
2299	Canton	<null>
4773	Medford	5083847566
3338	Canton	6179238844
2246	Medford	5083457789
1034	Framingham	6172251178
2424	Natick	5083347700
3767	Natick	5082844094
2898	Medford	6173458860
3449	Concord	5082684508
3082	Camden	5089273644
3341	Newton	6179658099
4660	Framingham	5092344620
2209	Brookline	6175563331
2894	Taunton	5088449008
4001	Natick	5082649956
5090	Canton	5083389935
1765	Grover	5087394772
4456	Newton	6179660089
2145	Wilmington	5087486625
3991	Taunton	5081840883
3778	Medford	6173450099
4358	Wilmington	5087488011
4962	Boston	6177732280
2180	Medford	5083145366
2106	Medford	5083346364
3222	Brookline	6176635520
4002	Wilmington	5088477701
2437	Boston	6179264105
2096	Brookline	6175553643
2004	Medford	5089253998
5103	Brookline	6176600684
5008	Brookline	6176642209
4321	Grover	5087463300
2598	Camden	<null>
3764	Brookline	6179458377
2461	Medford	5083873664
2448	Natick	5082844736
1003	Boston	6173295757
1234	Brookline	6176646602
2466	Medford	5089487709
4027	Natick	5089445386
2174	Brookline	6176633854
2781	Stoneham	6173286008
3704	Dedham	6177739440
4008	Brookline	<null>
3841	Boston	6173298763
3433	Newton	6178653440
3288	Newton	6179654443
4703	Brookline	6176648290
3294	Brookline	6175567551
3118	Canton	5083766984
3769	Grover	5084850432

52 rows processed

If your results do not match what you see above, check “Exercise 4-12 answer” on page B-11 for the correct SQL syntax. Remember, result tables may be shortened in this guide.

Compare the result of the first SELECT statement with the result of the second.

The first SELECT statement gives a more limited list because an employee must *both* live in one of the three communities *and* have a telephone in order to be included in the list.

The second SELECT statement results in a longer list because an employee must *either* live in one of the three communities *or* have a telephone (and live anywhere) in order to be included. Thus, the result table lists everyone who has a telephone and everyone who lives in the three communities.

Try one with both AND and OR: You can have multiple predicates, connecting them with either AND or OR.

The Human Resources department needs a list of employees who have a telephone and live in Brookline and all employees who live in Boston regardless of whether they have a telephone.

To produce this list, enter:

```
select emp_id, city, phone
       from employee
       where phone is not null and city = 'Brookline'
          or city = 'Boston';
```

The result looks like this:

EMP_ID	CITY	PHONE
3411	Boston	<null>
2010	Boston	<null>
2209	Brookline	6175563331
4962	Boston	6177732280
3222	Brookline	6176635520
2437	Boston	6179264105
2096	Brookline	6175553643
5103	Brookline	6176600684
5008	Brookline	6176642209
3764	Brookline	6179458377
1003	Boston	6173295757
1234	Brookline	6176646602
2174	Brookline	6176633854
3841	Boston	6173298763
4703	Brookline	6176648290
3294	Brookline	6175567551

16 rows processed

Using parentheses: The default order of evaluation is AND before OR. You use parentheses to override the default order of evaluation. Multiple search conditions enclosed in parentheses are evaluated as a single search condition.

How it's done: If the Human Resources department wants a list of employees living in Brookline or Boston who have a telephone, you would insert parentheses to group the Brookline and Boston predicates. The parentheses specify that you want the OR portion of the clause to be evaluated first:

```
select emp_id, city, phone
       from employee
       where phone is not null
          and (city = 'Brookline' or city = 'Boston');
```

The result looks like this:

EMP_ID	CITY	PHONE
2209	Brookline	6175563331
4962	Boston	6177732280
3222	Brookline	6176635520
2437	Boston	6179264105
2096	Brookline	6175553643
5103	Brookline	6176600684
5008	Brookline	6176642209
3764	Brookline	6179458377
1003	Boston	6173295757
1234	Brookline	6176646602
2174	Brookline	6176633854
3841	Boston	6173298763
4703	Brookline	6176648290
3294	Brookline	6175567551

14 rows processed

Compare this result with the result from the previous statement. The previous `SELECT` statement without parentheses listed employees who have a telephone and who also live in Brookline as well as employees who live in Boston whether or not they have a telephone.

The second `SELECT` statement with parentheses listed employees who live in either Brookline or Boston and who have a telephone, no matter which community they live in.

When you create a complex combination of predicates as in the last example, use parentheses to group predicates and establish the order of evaluation.

The placement of parentheses: You've just seen that using parentheses can make a difference in the order in which the predicates are evaluated.

Look at these two `SELECT` statements. They are exactly the same except for the placement of the parentheses.

Enter both `SELECT` statements and compare the results:

- ```
select emp_id, phone, city, dept_id
 from employee
 where phone is not null
 or (city = 'Boston' and dept_id = 5200);
```
- ```
select emp_id, phone, city, dept_id
       from employee
       where (phone is not null or city = 'Boston')
          and dept_id = 5200;
```

Now take all parentheses out of the request.

How are the results different?

The first SELECT statement specifies that the employee must *either* live in Boston and work in department 5200 *or* have a telephone in order to be on the list.

The second SELECT statement specifies that the employee must either have a telephone or live in Boston. In either case, the employee must also work in department 5200 in order to be placed on the list.

If you take out all the parentheses, SQL evaluates the conditions in the same order as for the first SELECT statement.

4.8 Review

Choose the correct answers for the questions below. More than one answer can apply for each question.

1. The clause that allows the user to specify a search condition that filters the rows to be selected is:
 - a. The WHERE clause
 - b. The FROM clause
 - c. The SELECT clause
 - d. The AS clause
2. What are the components of the WHERE clause?
 - a. The keyword FROM
 - b. The keyword WHERE
 - c. A search condition
 - d. A table name
3. You can compare a character column to a:
 - a. FROM clause
 - b. Mask
 - c. WHERE clause
 - d. Table name
4. Masks are used with:
 - a. The IN predicate
 - b. The BETWEEN predicate
 - c. The LIKE predicate
 - d. The FROM clause
5. Which of the following are mask characters?
 - a. #
 - b. _
 - c. *
 - d. %
6. The IS NULL predicate causes:
 - a. Retrieval of rows where a column contains the value zero
 - b. Retrieval of rows where a column contains no value
 - c. No retrieval of rows

d. No retrieval of rows where all columns contain zero

7. Parentheses are used to:

a. Set up the sequence of arithmetic evaluation

b. Set up the sequence of evaluation of AND and OR with multiple predicates

c. Set off table names

d. Set up alternate headings

To check your answers, see “Review answers” on page B-11.

4.9 Scenarios

Create the appropriate statements online to retrieve the needed data:

1. Periodically, a list is published giving divisions and their departments. A new department was recently added to division D09, so a new list for that division is needed. Use the DEPARTMENT table and show division code, department ID, and department name. Order by department ID.

Hint: DIV_CODE is a character column.

2. All Commonwealth Auto employees whose last names begin with L and M are due to have flu shots. The medical office needs to have the complete names of these individuals and the department to which each is assigned. Sort the list by last name. (Use the EMPLOYEE table.)

3. The Marketing department has a large project coming up and needs employees who have at least a medium level of competence (greater than 02) in skill 3333. Display employee ID and level of competence for each employee using the EXPERTISE table.

Hint: SKILL_LEVEL is a character column.

4. In order to identify employees involved in media projects, the Human Resources department needs a list of employees associated with a project ID that begins with P (indicating media-related). Order the list by employee ID. (Use the EMPLOYEE table to find this information.)

5. The budget group needs a list of employees who hold a position that pays less than \$25,000. Show employee ID and salary.

Hint: The POSITION table contains information about positions held in Commonwealth Auto.

To check your answers, see “Scenario answers” on page B-12.

Chapter 5. Using Aggregate Functions

5.1 About this chapter	5-3
5.2 Aggregate functions	5-4
5.2.1 Exercise 5-1	5-5
5.2.2 Exercise 5-2	5-5
5.2.3 Exercise 5-3	5-6
5.2.4 Exercise 5-4	5-7
5.2.5 Exercise 5-5	5-8
5.2.6 Exercise 5-6	5-8
5.2.7 Exercise 5-7	5-9
5.3 Eliminating duplicate rows	5-10
5.3.1 Exercise 5-8	5-10
5.4 Grouping information	5-11
5.4.1 Exercise 5-9	5-12
5.4.2 Exercise 5-10	5-12
5.5 Using HAVING	5-15
5.5.1 Exercise 5-11	5-15
5.5.2 Exercise 5-12	5-16
5.6 Renaming column headings	5-17
5.6.1 Exercise 5-13	5-17
5.7 Review	5-18
5.8 Scenarios	5-19

5.1 About this chapter

Goal: When you have completed this chapter, you will be able to use aggregate functions to count rows of data and to calculate averages, sums, maximums, and minimums for groups of data.

Summary: You can use the aggregate functions AVG, COUNT, MAX, MIN, and SUM to perform calculations within your SELECT statement to summarize information about groups of rows in a table.

5.2 Aggregate functions

There are five aggregate functions. Except for COUNT, these functions operate on a collection of values in one column of a table and produce a single result:

Function	Meaning
AVG	Returns the average of all values in the named column
COUNT	Counts the number of rows that satisfy a condition
MAX	Returns the highest value in the named column
MIN	Returns the lowest value in the named column
SUM	Returns the total of all values in the named column

How to use aggregate functions: To use an aggregate function, you specify the name of the function followed by the name of the column in parentheses, as in SUM(SALARY_AMOUNT).

You can use the aggregate functions AVG, MAX, MIN, and SUM with a column name (SALARY_AMOUNT) or, if you are using CA-IDMS/DB, you can use these functions with an arithmetic expression (SALARY_AMOUNT/52) as well.

Where to use aggregate functions

- Instead of a column name with SELECT
- In a HAVING clause as a value in a predicate

AVG: The aggregate function AVG calculates the **average** value of all rows in a specified column.

How it's done: The president of Commonwealth Auto wants to know the average salary for all employees in the company. To produce this information, use the POSITION table and enter:

```
select avg(salary_amount)
      from position;
```

The result looks like this:

(EXPR) ----- 51101.16
1 row processed

5.2.1 Exercise 5-1

Now you try it: Use the COVERAGE table to write a SELECT statement to display the average number of dependents for all employees in Commonwealth Auto. The Human Resources department needs this information for statistical purposes.

The result looks like this:

```

              (EXPR)
             -----
                1
1 row processed

```

If your results do not match what you see above, check “Exercise 5-1 answer” on page B-14 for the correct SQL syntax.

Data type: If the column being averaged has an integer data type, the result will be an integer (that is, a whole number). Refer to the SQL reference manual for your environment to learn how to convert an integer data type to a decimal data type in the result of your query.

5.2.2 Exercise 5-2

Using WHERE: Use the BENEFITS table to write a SELECT statement to display the average vacation accrued in fiscal year 1999 for all employees in Commonwealth Auto. The Human Resources department needs this information for statistical purposes.

The result looks like this:

```

              (EXPR)
             -----
            121.01
1 row processed

```

If your results do not match what you see above, check “Exercise 5-2 answer” on page B-14 for the correct SQL syntax.

COUNT: Use the aggregate function COUNT to **count** the number of rows in a table.

You use an asterisk in parentheses after COUNT when you want all rows to be counted. You use a column name in parentheses after COUNT when you want all rows with a value in that column to be counted.

How it's done: Human Resources needs a total count of employees working at Commonwealth Auto. To find this number, enter:

```
select count(*)
       from employee;
```

The result looks like this:

```
(EXPR)
-----
      55
1 row processed
```

Every row in the EMPLOYEE table was counted.

5.2.3 Exercise 5-3

Now you try it: The Human Resources department would like to know how many different types of skills there are in the company. Enter the appropriate statement to determine the total number of skills in the SKILL table.

The result looks like this:

```
(EXPR)
-----
      26
1 row processed
```

If your results do not match what you see above, check “Exercise 5-3 answer” on page B-14 for the correct SQL syntax.

Specifying a column name with COUNT: If you specify a column name with COUNT, only the rows containing a value in that column are counted.

How it's done: Human Resources has made another request. They would like to know how many employees have telephones in their homes. You can comply with this request by creating a SELECT statement using COUNT and the column name PHONE from the EMPLOYEE table:

```
select count(phone)
       from employee;
```

The result looks like this:

```

      (EXPR)
      -----
         49
1 row processed

```

There were fewer rows returned with this request than there were when you specified COUNT(*). This time, COUNT counted only the rows that had a telephone number. It did not count the rows that contain a null value for PHONE. When you used COUNT(*), *all* rows are counted.

5.2.4 Exercise 5-4

Using WHERE: How many employees in department 5200 have telephones? Add a WHERE clause to your previous statement to find out.

The result looks like this:

```

      (EXPR)
      -----
         4
1 row processed

```

If your results do not match what you see above, check “Exercise 5-4 answer” on page B-14 for the correct SQL syntax.

MAX: Use the aggregate function MAX (**maximum**) to determine the highest value in a specified column.

How it's done: The Human Resources department would like to know the highest salary in the company. Salaries are stored in the POSITION table. To show this information, enter:

```

select max(salary_amount)
       from position;

```

The result looks like this:

```

      (EXPR)
      -----
    146432.00
1 row processed

```

5.2.5 Exercise 5-5

Now you try it: Human Resources also needs to know the highest salary held for job 3333. Enter an appropriate `SELECT` statement using a `WHERE` clause.

The result looks like this:

```
(EXPR)
-----
30680.00
1 row processed
```

If your results do not match what you see above, check “Exercise 5-5 answer” on page B-14 for the correct SQL syntax.

MIN: Use the aggregate function `MIN` (**minimum**) to determine the lowest value in a specified column.

How it's done: Human Resources needs to determine the lowest salary in the company. To obtain this, enter:

```
select min(salary_amount)
       from position;
```

The result looks like this:

```
(EXPR)
-----
2200.00
1 row processed
```

5.2.6 Exercise 5-6

Now you try it: The Human Resources department is concerned that the company have a healthy group of employees. They need to see the smallest amount of sick time taken. The `BENEFITS` table contains this information.

The result looks like this:

```
(EXPR)
-----
0.00
1 row processed
```

If your results do not match what you see above, check “Exercise 5-6 answer” on page B-14 for the correct SQL syntax.

SUM: Use the aggregate function SUM to **total** numeric columns.

How it's done: The budget group in the Accounting department needs to allocate funds for next year's budget based on this year's salaries.

To obtain a sum of all salaries, enter:

```
select sum(salary_amount)
       from position;
```

The result looks like this:

```

              (EXPR)
              -----
              2555058.42

1 row processed
```

5.2.7 Exercise 5-7

Now you try it: Also as part of the budget process, the budget group needs to identify all vacation hours taken. (This information is in the BENEFITS table.)

Enter a statement to display this sum.

The result looks like this:

```

              (EXPR)
              -----
              17469.50

1 row processed
```

If your results do not match what you see above, check “Exercise 5-7 answer” on page B-14 for the correct SQL syntax.

Aggregate functions and null values: There may be null values in columns included in a calculation. Aggregate functions ignore rows where a null value is found.

5.3 Eliminating duplicate rows

You can eliminate duplicate rows when using SUM, AVG, and COUNT by using DISTINCT. This causes the duplicate rows to be eliminated before the aggregate function is applied.

Using DISTINCT: If you use DISTINCT, you must name a column explicitly; you cannot use an arithmetic expression.

How it's done: Put DISTINCT immediately before the column name.

Count the number of communities represented by the employees at Commonwealth Auto by entering:

```
select count(distinct city)
       from employee;
```

The result looks like this:

```
(EXPR)
-----
      13
1 row processed
```

5.3.1 Exercise 5-8

Now you try it: Count the number of different projects on which at least one person is working.

What table do you need to use?

You'll find this information in the CONSULTANT table.

The result looks like this:

```
(EXPR)
-----
      1
1 row processed
```

If your results do not match what you see above, check “Exercise 5-8 answer” on page B-14 for the correct SQL syntax.

5.4 Grouping information

You can use aggregate functions to display information for groups of rows rather than for a whole table. For example, the president wants to know the average salary for employees assigned to each job rather than the average salary for all employees in the whole company. The POSITION table, where the salary information is maintained, has more than one row for each job. In order to retrieve the information for the president, you need to group all rows with the same job ID and then find the average salary for that group.

Using GROUP BY: To summarize information for groups of rows, such as all employees who have the same job ID, use the GROUP BY clause. The GROUP BY clause indicates which columns contain values to be grouped together.

How it's done: To find the average salary for employees by job, use the GROUP BY clause and enter:

```
select job_id, avg(salary_amount)
       from position
       group by job_id;
```

The result looks like this:

JOB_ID	(EXPR)
2051	<null>
2053	<null>
2077	23672.56
3333	23130.05
4012	37546.80
4023	74776.00
4025	43888.00
4123	49921.76
4130	45241.94
4560	<null>
4666	85280.00
4700	53477.38
4734	55744.75
5110	56977.80
5555	54738.99
5890	53893.16
6004	110448.00
6011	94953.52
6021	111593.00
8001	117832.68
9001	146432.00

21 rows processed

The GROUP BY clause grouped the rows of data by job and then AVG took the average salary for each group. For example, nine salaries are averaged for JOB_ID 3333. Three jobs have null for an average salary because they have only hourly employees.

5.4.1 Exercise 5-9

Now you try it: For statistical purposes, you need to know the number of employees in each department. Enter a `SELECT` statement to retrieve this information from the `EMPLOYEE` table.

The result looks like this:

DEPT_ID	(EXPR)
1100	3
1110	2
1120	4
2200	5
2210	8
3510	2
3520	1
3530	2
4500	3
4600	9
5000	3
5100	2
5200	5
6200	6

14 rows processed

If your results do not match what you see above, check “Exercise 5-9 answer” on page B-14 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

5.4.2 Exercise 5-10

Try another: The budget group needs to identify the total salaries for each job in order to determine salary budgets for next year.

Using the `POSITION` table, enter a `SELECT` statement to show the sum of salaries for employees by job. Display job ID and sum for each job.

The result looks like this:

JOB_ID	(EXPR)
2051	<null>
2053	<null>
2077	118362.84
3333	208170.50
4012	150187.20
4023	74776.00
4025	43888.00
4123	49921.76
4130	45241.94
4560	<null>
4666	85280.00
4700	267386.90
4734	111489.50
5110	56977.80
5555	492650.98
5890	269465.80
6004	110448.00
6011	94953.52
6021	111593.00
8001	117832.68
9001	146432.00

21 rows processed

If your results do not match what you see above, check “Exercise 5-10 answer” on page B-14 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

Things to remember when using GROUP BY

- You can use GROUP BY with any aggregate function
- You can specify more than one column in the GROUP BY clause
- Columns listed in the SELECT statement that are not part of the calculation by an aggregate function must be identified in the GROUP BY clause

Using ORDER BY with aggregate functions: When you want to sort a result table based on an aggregate function, you must specify the aggregate function by number (or heading name) rather than specifying the function itself.

How it's done: Earlier you found the average salary for employees by job using GROUP BY. To sort this result by the average salary, enter:

```
select job_id, avg(salary_amount)
       from position
       group by job_id
       order by 2;
```

The result looks like this:

5.4 Grouping information

JOB_ID	(EXPR)
-----	-----
3333	23130.05
2077	23672.56
4012	37546.80
4025	43888.00
4130	45241.94
4123	49921.76
4700	53477.38
5890	53893.16
5555	54738.99
4734	55744.75
5110	56977.80
4023	74776.00
4666	85280.00
6011	94953.52
6004	110448.00
6021	111593.00
8001	117832.68
9001	146432.00
2053	<null>
4560	<null>
2051	<null>

21 rows processed

5.5 Using HAVING

You can add a search condition to use with an aggregate function.

The HAVING clause allows you to search for a particular condition within each group. HAVING takes the same predicates as WHERE. The clause must specify an aggregate function because it applies to summary rows only. You use a HAVING clause to eliminate groups from the result, just as you use a WHERE clause to eliminate rows.

You can have both a WHERE clause and a HAVING clause in your SELECT statement.

How it's done: The company is concerned that there are several departments with only a very few employees. To display those departments that have fewer than three employees, enter:

```
select dept_id, count(emp_id)
       from employee
       group by dept_id
       having count(emp_id) < 3;
```

The result looks like this:

DEPT_ID	(EXPR)
1110	2
3510	2
3520	1
3530	2
5100	2

5 rows processed

5.5.1 Exercise 5-11

Now you try it: Some Commonwealth Auto employees live in the same community. Since the company is encouraging car pooling, the car pooling group needs to know which communities have more than two people living there.

Enter a SELECT statement to list the name of the community and the number of employees living in that community. Show only those communities that have more than two employees. This information is contained in the EMPLOYEE table.

The result looks like this:

CITY	(EXPR)
----	-----
Boston	6
Brookline	11
Camden	3
Canton	4
Grover	4
Medford	9
Natick	5
Newton	4
Wilmington	3
9 rows processed	

If your results do not match what you see above, check “Exercise 5-11 answer” on page B-14 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

5.5.2 Exercise 5-12

Try another: The budget director has given you another request. She needs to see which jobs have an average salary greater than \$25,000. Use the POSITION table to write the appropriate SELECT statement using HAVING and display the job ID and the average salary amount.

The result looks like this:

JOB_ID	(EXPR)
-----	-----
4012	37546.80
4023	74776.00
4025	43888.00
4123	49921.76
4130	45241.94
4666	85280.00
4700	53477.38
4734	55744.75
5110	56977.80
5555	54738.99
5890	53893.16
6004	110448.00
6011	94953.52
6021	111593.00
8001	117832.68
9001	146432.00
16 rows processed	

If your results do not match what you see above, check “Exercise 5-12 answer” on page B-15 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

5.6 Renaming column headings

Remember that you can request new headings for the result table. This is particularly useful when you use aggregate functions because the default heading is not meaningful.

5.6.1 Exercise 5-13

Now you try one: Use the statement above, and rename the column with the aggregate function. Use the AS keyword, and make the heading "Average Salary".

The result looks like this:

JOB_ID	AVERAGE SALARY
4012	37546.80
4023	74776.00
4025	43888.00
4123	49921.76
4130	45241.94
4666	85280.00
4700	53477.38
4734	55744.75
5110	56977.80
5555	54738.99
5890	53893.16
6004	110448.00
6011	94953.52
6021	111593.00
8001	117832.68
9001	146432.00

16 rows processed

If your results do not match what you see above, check "Exercise 5-13 answer" on page B-15 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

5.7 Review

Fill in the blank with the correct term:

1. You use _____ to perform calculations within a SELECT statement.
2. An aggregate function can be used instead of a column name with _____ or in the _____ clause.
3. When the aggregate function AVG encounters a null value, it _____ the row.
4. The _____ clause acts as a search condition with an aggregate function.
5. You rename an aggregate function column heading by using _____ and the heading you want.

To check your answers, see “Review answers” on page B-15.

5.8 Scenarios

Create the appropriate statements online to retrieve the needed data:

1. In order to plan for the Christmas party for Commonwealth Auto, the Human Resources department needs a count of employees by department. (The EMPLOYEE table contains this information.)
2. As part of its salary research, the Human Resources department needs to know the minimum and maximum salaries being earned for each job ID in the company. (Use the POSITION table.)
3. Upper management needs to know how many subordinate employees there are for each manager in order to evaluate the span of control within the company. The EMPLOYEE table contains this information.
4. A project is coming up that requires project members having the skill ID 3333 (body work). The project leader needs to find out how many employees have a skill level greater than 02 for this skill to see whether he needs to hire consultants to staff the project. Keep in mind that the SKILL_LEVEL column contains character data. (Use the EXPERTISE table.)

Hint: Use a WHERE clause with more than one predicate.

5. The Human Resources department is conducting research into salaries. They have asked you for a report showing:
 - Job ID
 - Average salary by job
 - Minimum salary by job
 - Maximum salary by job

They need this report only for positions with a job ID less than 4000 (indicating training and clerical positions) where the average salary is less than \$25,000. Use the POSITION table and rename the column headings so that the report makes sense.

6. The training group is concerned that there are few people in the company who have certain crucial skills. They have asked you to give them a report listing the number of employees who have either a medium level of competence (02 or above) for skill 3333 (body work) or a high level of competence (04) for skill 4444 (assembly). The report should list a skill only if there are more than two employees that fit that category.

Hint: The column SKILL_LEVEL is a character column.

To check your answers, see “Scenario answers” on page B-15.

Chapter 6. Accessing Multiple Tables

6.1 About this chapter	6-3
6.2 What is a join operation?	6-4
6.2.1 Joining tables on common columns	6-4
6.2.1.1 Exercise 6-1	6-6
6.2.2 Qualifying a column name	6-7
6.2.3 Qualifying a table name	6-9
6.2.3.1 Exercise 6-2	6-9
6.2.4 Sorting the result	6-10
6.2.4.1 Exercise 6-3	6-10
6.2.5 Additional search criteria in a join	6-12
6.2.6 Exercise 6-4	6-12
6.2.7 Things to remember about joining tables	6-12
6.3 Joining a table to itself	6-14
6.4 Using UNION	6-16
6.5 Review	6-18
6.6 Scenarios	6-19

6.1 About this chapter

Goal: When you have completed this chapter, you will be able to create SQL statements that retrieve data from more than one table.

Summary: Until now, you have been retrieving data from only one table at a time. Often, however, the data you want resides in more than one table. For example, you may want information on the department an employee works in as well as the employee information itself. The **join** operation allows you to do this.

6.2 What is a join operation?

A join is a type of select in which you request data from more than one table.

Look at the table descriptions for EMPLOYEE and DEPARTMENT in Appendix C, “Table Descriptions.” The EMPLOYEE table carries employee information plus the employee's department ID. The *name* of the department is not carried in this table.

If you want to see more information about the department, you need to look at the DEPARTMENT table where you find the ID and name of the department. To display both employee and department information at the same time, you need to access both tables at once to **join** them.

Common columns: A **join** can occur when tables have a column in common. Each table must have at least one column that corresponds to a column in at least one other table in the join.

Usually these common columns are planned as part of the database design. In Chapter 1, “Relational Database Concepts” on page 1-1 you read about **foreign keys**, which are columns or combination of columns in one table corresponding to the primary key of another table. These are planned common columns:

EMP_ID	EMP_LNAME	EMP_FNAME	DEPT_ID
2096	CARLSON	THOMAS	4600
2437	THOMPSON	HENRY	4600
2598	JACOBS	MARY	5100

DEPT_ID	DEPT_NAME
5200	CORPORATE MARKETING
4600	MAINTENANCE
5100	BILLING

6.2.1 Joining tables on common columns

To join two tables, you must associate one or more columns in one table to one or more columns in a second table. The joining columns must:

- Exist in both tables
- Have data of equivalent type
- Appear in the WHERE clause

Normally, a row from one table is joined with a row from the other when the common columns contain equal values.

Statement used: The statement used to join tables is composed of:

- The columns to be displayed listed after SELECT
- A FROM clause identifying the tables from which data is being retrieved
- A WHERE clause indicating the column to be matched and any additional search conditions

Note: If you are joining tables from different schemas, you must preface the table name with the schema name.

In the WHERE clause, you specify a column name from one table, a comparison operator (usually =), and a column name from the other table.

How it's done: Periodically, the Human Resources department produces a list of the employees who head departments.

The ID of the employee who heads a particular department is found in the DEPARTMENT table. The employee's name is found in the EMPLOYEE table. You need to join these two tables to get all the information for the list.

To join the DEPARTMENT and EMPLOYEE tables based on the head of the department, you use the DEPT_HEAD_ID column in the DEPARTMENT table and the EMP_ID column in the EMPLOYEE table. The columns have different names, but both contain employee IDs.

To join the two tables, enter:

```
select emp_id, emp_lname, emp_fname, dept_name
       from employee, department
       where dept_head_id = emp_id;
```

In this statement, all the join columns have unique names. The SELECT statement specifies that you want to see employee ID, employee last name and first name, and name of department. This information is going to come from two different tables. The common columns are matched in the WHERE clause.

The result looks like this:

6.2 What is a join operation?

```
EMP_ID  EMP_LNAME      EMP_FNAME      DEPT_NAME
-----  -
2004    Johnson        Eleanor         PURCHASING - SERVICE
1003    Baldwin        James          LEASING - NEW CARS
2466    Bennett        Patricia       MIS
2010    Parker         Cora           SALES - NEW CARS
3769    Doneison       Julie          APPRAISAL NEW CARS
2466    Bennett        Patricia       CORPORATE ACCOUNTING
3222    Voltmer        Louise         HUMAN RESOURCES
2096    Carlson        Thomas        MAINTENANCE
2180    Albertini      Joan           SALES - USED CARS
2598    Jacobs         Mary           BILLING
2461    Anderson       Alice          CORPORATE ADMINISTRATION
2209    Smith          Michael        APPRAISAL - SERVICE
1003    Baldwin        James          LEGAL
3082    Brooks         John           APPRAISAL - USED CARS
2246    Hamel         Marylou        PURCHASING - USED CARS
2894    Griffin        William        CORPORATE MARKETING
1765    Alexander      David          PURCHASING - NEW CARS
17 rows processed
```

6.2.1.1 Exercise 6-1

Now you try it: The Human Resources department needs information about divisions. They have asked you for a list of division descriptions and division heads by name.

Enter a SELECT statement to display each division description and employee last and first name.

What tables do you need to join?

You need to join the DIVISION and EMPLOYEE tables.

What are the common columns?

DIV_HEAD_ID in the DIVISION table and EMP_ID in the EMPLOYEE table are the common columns.

The result looks like this:

```
DIV_CODE  DIV_NAME      EMP_ID  EMP_LNAME      EMP_FNAME
-----  -
D06       SERVICE       4321    Bradley        George
D04       NEW CARS      2010    Parker         Cora
D09       CORPORATE     1003    Baldwin        James
D02       USED CARS     2180    Albertini      Joan
4 rows processed
```

If your results do not match what you see above, check “Exercise 6-1 answer” on page B-19 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

Why include the join condition: Without the join condition in the WHERE clause, the request would return a huge table containing every possible row combination from the tables being joined. This type of join is called a Cartesian product. It is very inefficient and contains a great deal of redundant information.

6.2.2 Qualifying a column name

In some cases, the join columns from two tables have the same name. Then you must add a qualification to the column names to distinguish one name from the other.

How it's done: The EMPLOYEE table does not carry the department name, and the DEPARTMENT table does not carry the employee ID. If you want to see the names of the departments that the employees are associated with, you need to join the EMPLOYEE and DEPARTMENT tables. These two tables have a common column, DEPT_ID.

Because the column DEPT_ID has the same name in each table, you need to qualify each common column name with the name of its table. Enter:

```
select emp_id, department.dept_id, dept_name
       from department, employee
       where department.dept_id = employee.dept_id;
```

This statement specifies that the rows to be retrieved and joined from the EMPLOYEE and DEPARTMENT tables are those that have matching department IDs. If a department ID is present in the DEPARTMENT table but not in the EMPLOYEE table (as when a department has no employees), that row will not be returned.

The result looks like this:

6.2 What is a join operation?

```
EMP_ID  DEPT_ID  DEPT_NAME
-----  -
2299    4600    MAINTENANCE
3411    5200    CORPORATE MARKETING
4773    3510    APPRAISAL - USED CARS
2010    2210    SALES - NEW CARS
3338    1120    PURCHASING - SERVICE
2246    1100    PURCHASING - USED CARS
1034    4600    MAINTENANCE
2424    4600    MAINTENANCE
3767    2200    SALES - USED CARS
2898    1120    PURCHASING - SERVICE
3449    5000    CORPORATE ACCOUNTING
3082    3510    APPRAISAL - USED CARS
3341    3530    APPRAISAL - SERVICE
3199    4600    MAINTENANCE
4660    2200    SALES - USED CARS
2209    3530    APPRAISAL - SERVICE
2894    5200    CORPORATE MARKETING
4001    2210    SALES - NEW CARS
5090    2210    SALES - NEW CARS
1765    1110    PURCHASING - NEW CARS
4456    4600    MAINTENANCE
2145    5200    CORPORATE MARKETING
3991    2210    SALES - NEW CARS
3778    5100    BILLING
4358    5200    CORPORATE MARKETING
4962    2210    SALES - NEW CARS
2180    2200    SALES - USED CARS
2106    1110    PURCHASING - NEW CARS
3222    4500    HUMAN RESOURCES
4002    6200    CORPORATE ADMINISTRATION
2437    4600    MAINTENANCE
2096    4600    MAINTENANCE
2004    1120    PURCHASING - SERVICE
5103    5000    CORPORATE ACCOUNTING
5008    1100    PURCHASING - USED CARS
4321    6200    CORPORATE ADMINISTRATION
2598    5100    BILLING
3764    2210    SALES - NEW CARS
2461    6200    CORPORATE ADMINISTRATION
2448    2200    SALES - USED CARS
1003    6200    CORPORATE ADMINISTRATION
1234    6200    CORPORATE ADMINISTRATION
2466    5000    CORPORATE ACCOUNTING
4027    2210    SALES - NEW CARS
2174    4500    HUMAN RESOURCES
2781    5200    CORPORATE MARKETING
3704    2200    SALES - USED CARS
4008    2210    SALES - NEW CARS
3841    6200    CORPORATE ADMINISTRATION
3433    4600    MAINTENANCE
3288    4600    MAINTENANCE
4703    1100    PURCHASING - USED CARS
3294    1120    PURCHASING - SERVICE
3118    4500    HUMAN RESOURCES
3769    3520    APPRAISAL NEW CARS
```

55 rows processed

Does it matter which of the two department ID columns you choose to display?

No. The values in each of the two matching columns is the same.

6.2.3 Qualifying a table name

In some cases, the tables are in different schemas. Then you must add a qualification to the table name to specify which schema the table is associated with.

How it's done: The EMPLOYEE table doesn't contain project id and description information, and the PROJECT table doesn't carry the project leader's first and last name. If you want to see the projects and their respective project leaders, you must join the EMPLOYEE and PROJECT tables using the PROJ_LEADER_ID and EMP_ID columns. However, the EMPLOYEE table is assigned to the DEMOEMPL schema and the PROJECT table is assigned to the DEMOPROJ schema. To access data from both schemas, you must qualify the table names with the schema name.

Enter:

```
select emp_id, emp_lname, emp_fname, proj_desc from
       demoempl.employee, demoproj.project
       where emp_id=proj_leader_id;
```

EMP_ID	EMP_LNAME	EMP_FNAME	PROJ_DESC
3411	Williams	Catherine	TV ads - WTVK
3411	Williams	Catherine	New brand research
2894	Griffin	William	Consumer study
4358	Robinson	Judith	Service study
2466	Bennett	Patricia	Systems analysis

5 rows processed

6.2.3.1 Exercise 6-2

Now you try it: You have to give the Human Resources department a list of employees and the skills each has. However, it is easier to read this report if the employees' names are listed as well.

Enter a SELECT statement to list employee ID, last name, first name, and skill ID using the EMPLOYEE and EXPERTISE tables. Qualify the column name that is the same in both tables. The EXPERTISE table is in DEMOPROJ and the EMPLOYEE table is in DEMOEMPL.

The result looks like this:

EMP_ID	EMP_LNAME	EMP_FNAME	SKILL_ID
2174	Zander	Jonathan	4430
3118	Wooding	Alan	5180
5090	Wills	Stephen	7000
3411	Williams	Catherine	5500
3991	Wilkins	Fred	7000
2424	Wilder	Ronald	6470
2106	Widman	Susan	6770
4962	White	Peter	5130
3338	White	Mark	6770
3222	Voltmer	Louise	1000
3222	Voltmer	Louise	4430
2781	Thurston	Joseph	5420
2781	Thurston	Joseph	5430
4456	Thompson	Thomas	3065
4456	Thompson	Thomas	6670
4001	Thompson	Jason	7000
2437	Thompson	Henry	3333
2437	Thompson	Henry	4444
3449	Taylor	Cynthia	5200
2209	Smith	Michael	5309
3341	Smith	Carl	5309
3288	Sampson	Ralph	6670
3288	Sampson	Ralph	6650
3288	Sampson	Ralph	3333
4002	Roy	Linda	4410
4002	Roy	Linda	4370
4358	Robinson	Judith	5500
2010	Parker	Cora	7000
3764	Park	Deborah	7000
3704	Moore	Richard	7000
1234	Mills	Thomas	1000
4660	MacGregor	Bruce	7000
2448	Lynn	David	7000
3767	Lowe	Frank	7000
2004	Johnson	Eleanor	6770
3294	Johnson	Carolyn	6770
2598	Jacobs	Mary	6666
2246	Hamel	Marylou	1000
2246	Hamel	Marylou	6670
4703	Halloran	Martin	5130
4703	Halloran	Martin	4250
2894	Griffin	William	5500
2894	Griffin	William	1000
1034	Gallway	James	6470
5008	Fordman	Timothy	6770
3778	Ferndale	Jane	6666
3778	Ferndale	Jane	5200
5103	Ferguson	Adele	5200
3769	Donelson	Julie	5309
4773	Dexter	Janice	5309
3841	Cromwell	Michelle	4370
.			
.			
.			

69 rows processed

If your results do not match what you see above, check “Exercise 6-3 answer” on page B-19 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

6.2.5 Additional search criteria in a join

Remember that you specify the columns to be joined in the WHERE clause. You can have additional search criteria in the WHERE clause as well by using AND and OR.

6.2.6 Exercise 6-4

Now you try it: Human Resources wants a list of managers in charge of projects. You need to create a list of those managers with their IDs, last and first names.

What tables will you join?

You'll join the EMPLOYEE and CONSULTANT tables.

What are the join columns?

EMP_ID in EMPLOYEE and MANAGER_ID in CONSULTANT are the join columns.

Do you need to qualify any column name?

Yes, both tables have MANAGER_ID columns.

What additional search criteria do you need?

You need to specify that you only want each manager listed once.

The result looks like this:

MANAGER_ID	EMP_LNAME	EMP_FNAME
1003	Baldwin	James
2466	Bennett	Patricia

2 rows processed

If your results do not match what you see above, check “Exercise 6-4 answer” on page B-19 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

6.2.7 Things to remember about joining tables

When you join tables:

- You are combining rows of data from two or more tables to form one result table
- The join is based on common columns
- If a column name in the SELECT statement has the same name in two or more tables, it must be prefixed to specify the table

- The FROM clause contains the names of all the tables you are joining
- You should use a WHERE clause to limit the result table

6.3 Joining a table to itself

You join a table to itself when two rows in a table contain information that you want to combine together. This is called a **reflexive join**.

How it's done: The Human Resources department needs to identify Catherine William's manager. The EMPLOYEE table contains the employee ID and the manager ID. To find the name of Catherine William's manager, first you find Catherine William's employee ID (3411) in the EMP_ID column. The row containing Catherine William's employee ID also contains her manager's employee ID (2894) in the MANAGER_ID column. Now you look in the EMPLOYEE table again to find the manager's ID (2894) in the employee ID column. You'll find the manager's name in this row:

EMPLOYEE

EMP_ID	EMP_LNAME	MANAGER_ID
2096	CARLSON	4321
3411	WILLIAMS	2894
2894	GRIFFIN	1003

↓

EMP_ID	EMP_LNAME	MANAGER_ID	EMP_ID	EMP_LNAME	MANAGER_ID
3411	WILLIAMS	2894	2894	GRIFFIN	1003

To access this information, enter:

```
select mgr.emp_lname as "Manager",
       sub.emp_lname as "Subordinate"
  from employee mgr, employee sub
 where mgr.emp_id = sub.manager_id
        and sub.emp_id = 3411;
```

The result looks like this:

Manager	Subordinate
-----	-----
Griffin	Williams
1 row processed	

Things to remember about a reflexive join

- You assign aliases so that the manager's employee ID can be distinguished from the subordinate's employee ID

- You name the table twice in the FROM clause and give each reference an alias
- You compare the two columns that share the same type of information in the WHERE clause (MGR.EMP_ID = SUB.MANAGER_ID)

6.4 Using UNION

You can use UNION to append the rows returned by one set of selection criteria to the rows returned by another set.

Appending is different from joining. To join tables, you merge selected *columns* of one table with selected columns of another table. To append tables, you combine selected *rows* of one table to selected rows of another table.

How it's done: To combine the CONSULTANT and EMPLOYEE tables to get a complete list of all people on the payroll, enter:

```
select con_id, con_lname, con_fname
       from consultant
       union
       select emp_id, emp_lname, emp_fname
       from employee;
```

This statement adds rows from the EMPLOYEE table to the rows in the CONSULTANT table.

The result looks like this:

EMP_ID	EMP_LNAME	EMP_FNAME
-----	-----	-----
		.
		.
		.
4321	Bradley	George
4358	Robinson	Judith
4456	Thompson	Thomas
4660	MacGregor	Bruce
4703	Halloran	Martin
4773	Dexter	Janice
4962	White	Peter
5008	Fordman	Timothy
5090	Wills	Stephen
5103	Ferguson	Adele
9000	Legato	James
9388	Candido	Linda
9439	Miller	Charles
9443	Jones	Diane
59 rows	processed	

Things to remember about UNION

- UNION combines rows from separate SELECT statements and removes duplicate rows
- The columns you pair must match in length and data type and in allowing or not allowing null values
- You must name the same number of columns in each of the SELECT statements
- Column names do not have to match

- If you want to sort the result table, use a single ORDER BY clause after the last SELECT statement
- You must use column numbers if you want to sort the result table
- The column names from the second table are used as column headings
- You can specify column headings using AS on the second table

Adding rows selectively: By adding WHERE clauses, you can use UNION to selectively add rows from one or more tables to another table.

How it's done: To see information on both employees and consultants working in a particular department, enter:

```
select con_id, con_lname, con_fname
       from consultant
       where dept_id = 5200
       union
       select emp_id, emp_lname, emp_fname
       from employee
       where dept_id = 5200;
```

This statement adds selected rows from the EMPLOYEE table to selected rows in the CONSULTANT table.

The result looks like this:

EMP_ID	EMP_LNAME	EMP_FNAME
2145	Catlin	Martin
2781	Thurston	Joseph
2894	Griffin	William
3411	Williams	Catherine
4358	Robinson	Judith
9388	Candido	Linda
9443	Jones	Diane

7 rows processed

6.5 Review

Match each statement on the left with a term on the right. There will be one term left over.

Description	Term
1. Needed to join two or more tables	a. The WHERE clause
2. Resolves the problem of joining table columns that have the same name	b. Common columns
3. Where the joining is specified	c. The UNION clause
4. Where an alias is identified	d. The FROM clause
5. Used to append one table to another	e. The SELECT clause
	f. Aliases

To check your answers, see “Review answers” on page B-19.

6.6 Scenarios

Create the appropriate statements online to retrieve the needed data:

1. Management would like to see which employees are involved in which projects. Write a SELECT statement to retrieve this information by joining the EMPLOYEE and PROJECT tables that contain the data. Display the information by project description.
2. The Human Resources department needs a list of employees and their remaining vacation time. This information is contained in the EMPLOYEE and BENEFITS tables. Display employee ID and last name as well as the vacation time remaining for fiscal year 2000. Order your report by employee ID.
3. More statistics are being gathered on vacation hours. You have been asked to produce a report of average vacation hours taken for each department. Display department ID and average vacation taken for fiscal year 1999. Order the report by department ID.
4. The budget committee needs a list of job titles, names of employees holding those jobs, and current salaries of those employees. They are interested only in jobs offering salaries of more than \$55,000. Order your list by job title and include the job ID.

Hint: You need to join three tables to get this information. Specify the three tables in the FROM clause. Join two tables at a time in the WHERE clause. Use aliases to qualify column names.

5. Employee 2004 has just had a review and is due to get a pay increase. The increase is stored as REVIEW_PERCENT in the BENEFITS table. Employee 2004's manager has asked you to show her how much the increase is in dollar amount. To get this information, you need to multiply the current salary for fiscal year 2000 by the review percent. Show the employee ID, current salary, percent increase, and increase as a dollar amount.

Hint: You need to perform a calculation involving columns from two different tables.

To check your answers, see “Scenario answers” on page B-20.

Chapter 7. Nesting SELECT Statements

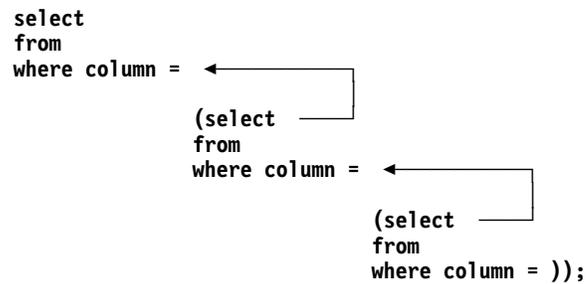
7.1 About this chapter	7-3
7.2 SELECT statement in a WHERE clause	7-4
7.3 Using a subquery with IN	7-5
7.3.1 Exercise 7-1	7-5
7.4 Using an aggregate function in a nested SELECT statement	7-7
7.4.1 Exercise 7-2	7-7
7.4.1.1 Exercise 7-2B	7-8
7.5 Using EXISTS	7-10
7.5.1 Exercise 7-3	7-11
7.6 Things to remember about subqueries	7-14
7.7 Review	7-15
7.8 Scenarios	7-16

7.1 About this chapter

Goal: When you have completed this chapter, you will be able to nest a SELECT statement within another SELECT statement to retrieve specified data.

Summary: An SQL request that is nested inside another SELECT statement is called a **subquery**. The subquery returns a set of values for use in the outer SELECT statement:

```
select
from
where column = (select
from
where column = (select
from
where column = ));
```



You nest SELECT statements when you want to use data from one table as part of the criteria of another table. A subquery is often used in conjunction with predicates IN and EXISTS.

7.2 SELECT statement in a WHERE clause

You use a `SELECT` statement in a `WHERE` clause to create a result table that limits the rows that can be retrieved by the outer `SELECT` statement.

Generally, the `SELECT` statement nested within a `WHERE` clause can return only one column.

7.3 Using a subquery with IN

Often, you want to retrieve rows from a table provided that values in a particular column are in another table. You can use IN as you did in Chapter 4. The nested SELECT statement provides the list that follows IN.

How it's done: You might want the last names and telephones of employees who are department heads. To retrieve the information from the EMPLOYEE table, first determine the employees that are the heads of departments. You can do this through a subquery. You must enclose the subquery in parentheses. Enter:

```
select emp_lname, phone, dept_id
       from employee
       where emp_id in
             (select dept_head_id
              from department);
```

The subquery first retrieves all the employee IDs of department heads from the DEPARTMENT table. The outer SELECT statement then uses this list to retrieve the last name, phone, and department ID of these employees from the EMPLOYEE table.

The result looks like this:

EMP_LNAME	PHONE	DEPT_ID
Albertini	5083145366	2200
Alexander	5087394772	1110
Anderson	5083873664	6200
Baldwin	6173295757	6200
Bennett	5089487709	5000
Brooks	5089273644	3510
Carlson	6175553643	4600
Donelson	5084850432	3520
Griffin	5088449008	5200
Hamel	5083457789	1100
Jacobs	<null>	5100
Johnson	5089253998	1120
Parker	<null>	2210
Smith	6175563331	3530
Voltmer	6176635520	4500

15 rows processed

7.3.1 Exercise 7-1

Now you try it: The Human Resources department needs to find out which departments have employees with more than 80 hours of vacation remaining for fiscal year 1999 so that the department head can be notified.

Enter a SELECT statement using a subquery to identify the IDs of those departments.

What tables are involved?

You need to access the BENEFITS table first to find out about remaining vacation hours. Then access the EMPLOYEE table to find the department ID.

You can use DISTINCT to eliminate duplicates.

The result looks like this:

```
DEPT_ID
-----
  4600
  6200

2 rows processed
```

If your results do not match what you see above, check “Exercise 7-1 answer” on page B-24 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

7.4 Using an aggregate function in a nested SELECT statement

You can use an aggregate function in a nested SELECT statement when you want to compare a value in a table with another value derived through an aggregate function.

How it's done: You want to see job IDs with current salaries that are higher than the average salary for all jobs. To do this, you use the POSITION table and enter:

```
select job_id, salary_amount
       from position
       where salary_amount >
             (select avg(salary_amount)
              from position);
```

This statement first finds the average salary for all jobs in the POSITION table and then looks at all jobs to see which exceed that average.

The result looks like this:

JOB_ID	SALARY_AMOUNT
4734	53665.00
5555	76440.00
4700	59488.00
4023	74776.00
5890	68016.00
5890	66144.00
6021	111593.00
4734	57824.50
5555	76961.00
6004	110448.00
4666	85280.00
4700	59280.00
5110	56977.80
5555	54184.00
5555	70720.00
8001	117832.68
9001	146432.00
6011	94953.52
4700	53665.56

19 rows processed

Enter the subquery SELECT statement alone to check these results by looking at the average salary itself.

7.4.1 Exercise 7-2

Now you try it: As part of the Human Resources department's research on insurance claims, they need a list of employees who have more than the average number of dependents. To get this information, you need access to the COVERAGE table. Your report should display employee ID and number of dependents.

The result looks like this:

EMP_ID	NUM_DEPENDENTS
3411	3
3411	3
3338	2
2246	2
2246	2
3767	2
3767	2
3199	2
3199	2
2894	3
2894	3
5090	3
1765	2
1765	2
3991	5
3991	5
3991	5
4962	4
3222	2
2437	2
2096	3
2096	3
5008	2
5008	2
2448	3
1234	5
1003	3
1003	3
2781	2
3704	3

30 rows processed

If your results do not match what you see above, check “Exercise 7-2 answer” on page B-24 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

7.4.1.1 Exercise 7-2B

Human Resources has requested you remove duplicate employees from the report.

EMP_ID	NUM_DEPENDENTS
1003	3
1234	5
1765	2
2096	3
2246	2
2437	2
2448	3
2781	2
2894	3
3199	2
3222	2
3338	2
3411	3
3704	3
3767	2
3991	5
4962	4
5008	2
5090	3

19 rows processed

If your results do not match what you see above, check “Exercise 7-2b answer” on page B-24 for the correct SQL syntax. Remember result tables may be shortened in this guide.

7.5 Using EXISTS

When you want to retrieve rows from a table based on the existence of rows in another table, use the EXISTS predicate. The EXISTS predicate includes a subquery. If rows in a table meet the selection criteria in the subquery, the outer SELECT statement proceeds. With the EXISTS predicate, you usually use * rather than a column name with SELECT in the subquery for simplicity.

How it's done: You want to retrieve the names of employees who have a certain level of a certain skill. You need to access both the EXPERTISE and EMPLOYEE tables to do this. Enter:

```
select emp_lname, emp_fname
       from employee
       where exists
         (select *
          from expertise
          where skill_id = 4444
                and skill_level = '04'
                and employee.emp_id = expertise.emp_id);
```

The outer SELECT statement looks at the first row in the EMPLOYEE table and passes the employee ID to the subquery. The subquery then evaluates this row by checking the employee ID against the criteria in the WHERE clause.

The outer SELECT statement and the subquery are connected by comparing common columns in the WHERE clause of the subquery.

The result looks like this:

EMP_LNAME	EMP_FNAME
Thompson	Henry

1 row processed

Here's another: If you want to list all the jobs in which an employee earns more than \$65,000, enter:

```
select job_id, job_title
       from job
       where exists
         (select *
          from position
          where salary_amount > 65000
                and "position".job_id = job.job_id);
```

Notes:

POSITION is the table name **and** an SQL keyword; therefore, when the POSITION table name is used as an identifier, it must be enclosed in double quotation marks.

As an alternative, you can use an alias for the table name. For example:

```
select job_id, job_title
       from job
       where exists
           (select *
            from position p
            where salary_amount > 65000
              and p.job_id = job.job_id);
```

The outer SELECT statement looks at the first row of the JOB table and passes the job ID to the subquery. If the row meets the selection criteria set up by the WHERE clause in the subquery, the row is displayed.

You need to qualify the column names in this example because the name, JOB_ID, is the same in both tables.

The result looks like this:

JOB_ID	JOB_TITLE
8001	Vice President
9001	President
6011	Manager - Acctng
4666	Sr Mechanic
5555	Salesperson
4023	Accountant
6004	Manager - HR
5890	Appraisal Spec
6021	Manager - Mktng

9 rows processed

7.5.1 Exercise 7-3

Now you try it: The budget group needs to know the department IDs of all departments where an employee earns more than \$50,000. Enter a SELECT statement that will show this information using the POSITION and EMPLOYEE tables.

Hint: Use DISTINCT to eliminate duplicates.

The result looks like this:

```
DEPT_ID
-----
1100
1120
2200
2210
3510
3530
4500
4600
5000
5200
6200

11 rows processed
```

If your results do not match what you see above, check “Exercise 7-3 answer” on page B-24 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

Using NOT EXISTS: You may also want to retrieve information from a table provided that *no* rows in another table meet the selection criteria.

For example, you might want to look for possible job openings by finding jobs that have no associated employee. To retrieve this information from the JOB table, first determine which jobs do not exist in the POSITION table. Enter:

```
select job_id, job_title
       from job
       where not exists
           (select * from position
            where "position".job_id = job.job_id);
```

Notes:

POSITION is the table name **and** an SQL keyword; therefore, when the POSITION table name is used as an identifier, it must be enclosed in double quotation marks.

As an alternative, you can use an alias for the table name. For example:

```
select job_id, job_title
       from job
       where not exists
           (select * from position p
            where p.job_id = job.job_id);
```

The result looks like this:

```
JOB_ID  JOB_TITLE
-----  -----
3051    Data Entry Clerk
3029    Computer Operator
5111    CUST SER REP
2055    PAYROLL CLERK

4 rows processed
```

7.6 Things to remember about subqueries

- You usually use `SELECT *` in a subquery in an `EXISTS` predicate
- You must enclose the nested `SELECT` statement in parentheses
- You join the outer `SELECT` statement and the subquery when using the `EXISTS` predicate by comparing columns in the `WHERE` clause of the subquery
- You can use aliases to qualify column names

7.7 Review

Fill in the blanks with the appropriate words or phrases:

1. A nested SELECT statement is also known as a _____.
2. A subquery is located in a _____ clause.
3. A subquery must be enclosed in _____.
4. You use the _____ predicate to retrieve rows based on the existence of rows in another table.
5. When using an EXISTS predicate, the outer SELECT statement and the subquery are linked by matching _____ in the WHERE clause in the subquery.
6. You can use an asterisk (*) in the subquery if you are using the _____ keyword.

To check your answers, see “Review answers” on page B-24.

7.8 Scenarios

Create the appropriate statements online to retrieve the needed data:

1. For tax purposes, the Accounting department needs to keep track of all jobs for which employees earn more than \$65,000. A list of job titles is sufficient. (Use the JOB and POSITION tables.)
2. Upper management is concerned about the equality of salaries within Commonwealth Auto. They need to have a list by name of all jobs for which at least one employee earns less than \$35,000. (Use the JOB and POSITION tables.)
3. Over the years, lots of department information has been added to the database. The Human Resources department is responsible for this portion of the database and knows that there are some departments still listed for which there are no longer any associated employees. They have asked you for a list showing these departments. Order the list by department ID. (Use the DEPARTMENT and EMPLOYEE tables.)

To check your answers, see “Scenario answers” on page B-25.

Chapter 8. Updating a Table

- 8.1 About this chapter 8-3
- 8.2 Inserting data into a table 8-4
 - 8.2.1 Exercise 8-1 8-4
 - 8.2.2 Exercise 8-2 8-6
 - 8.2.3 Exercise 8-3 8-6
- 8.3 Modifying data in a table with SET 8-8
 - 8.3.1 Exercise 8-4 8-8
 - 8.3.2 Exercise 8-5 8-10
 - 8.3.3 Exercise 8-6 8-10
 - 8.3.4 Exercise 8-7 8-11
- 8.4 Removing data from a table 8-12
 - 8.4.1 Exercise 8-8 8-12
 - 8.4.2 Exercise 8-9 8-13
- 8.5 Review 8-14

8.1 About this chapter

Goal: When you have completed this chapter, you will be able to store rows of data in a table and change and delete existing data in a table.

Summary: Up to this time, you have been retrieving data that exists in a table in the database. Now you are going to add, modify, and delete rows of data in a table. The statements you will use are:

- **INSERT**, to add a row of data to a table
- **UPDATE**, to change the values in one or more columns in all rows of a table or in rows that satisfy a search condition
- **DELETE**, to remove one or more rows from a table

8.2 Inserting data into a table

To add complete new rows of data to an existing table, use the INSERT statement and specify the values you want to add.

How it's done without column names: Suppose the company sets up a new department, the Audit department, and you need to add this information to the DEPARTMENT table. To do this, enter:

```
insert into department
  values (4040, 1234, 'D09', 'Audit');
```

The statement above adds the row into the table with the other department information.

Things to remember when using INSERT

- You can add only one row of values with one INSERT statement.
- Use INSERT INTO to identify the table to which you are going to add a new row
- Use the VALUES clause to identify the column values
- Enclose the column values in parentheses, separating them with commas
- Enclose values for character string data in single quotation marks
- The order of the column values must match the order of the columns defined for the table.
- Refer to the SQL reference manual for your environment to determine how to enter dates, money, and other types of data

Using the keyword NULL: If you do not have data available for a particular column, you can insert the keyword NULL as a place holder if the column has been defined to allow null values.

8.2.1 Exercise 8-1

Now you try it: Add another department to Commonwealth Auto. The only values you have right now are the department ID, 6060, the department name, Claims, and the division code, D09. The department name and division code columns contain character data. You do not know the ID of the head of the department.

Enter a SELECT statement to display all departments in order by department id and confirm your addition.

The result looks like this:

DEPT_ID	DEPT_HEAD_ID	DIV_CODE	DEPT_NAME
1100	2246	D02	PURCHASING - USED CARS
1110	1765	D04	PURCHASING - NEW CARS
1120	2004	D06	PURCHASING - SERVICE
2200	2180	D02	SALES - USED CARS
2210	2010	D04	SALES - NEW CARS
3510	3082	D02	APPRAISAL - USED CARS
3520	3769	D04	APPRAISAL NEW CARS
3530	2209	D06	APPRAISAL - SERVICE
4040	1234	D09	Audit
4200	1003	D04	LEASING - NEW CARS
4500	3222	D09	HUMAN RESOURCES
4600	2096	D06	MAINTENANCE
4900	2466	D09	MIS
5000	2466	D09	CORPORATE ACCOUNTING
5100	2598	D06	BILLING
5200	2894	D09	CORPORATE MARKETING
6000	1003	D09	LEGAL
6060	<null>	D09	Claims
6200	2461	D09	CORPORATE ADMINISTRATION

19 rows processed

If your results do not match what you see above, check “Exercise 8-1 answer” on page B-27 for the correct SQL syntax. Remember that result tables may be shortened in this guide.

Using INSERT with column names: If you want to insert only one or a few columns in a row, specify the column names.

How it's done: Suppose you want to add yet another department to the company, but you have only a department ID, department name, and division code. Enter:

```
insert into department (dept_id, dept_name, div_code)
values (5050, 'Research', 'D09');
```

The column names are in parentheses and separated by commas.

The values must be given in the correct column order. You can use the word NULL when you don't have a value in a column as long as the column accepts null values.

If you do not specify a column and a value for that column, a null value will be inserted for you. If the column does not accept null values, the insert is rejected.

Enter a SELECT statement to display the DEPARTMENT table in Department id order to confirm the insertion.

The result looks like this:

DEPT_ID	DEPT_HEAD_ID	DIV_CODE	DEPT_NAME
1100	2246	D02	PURCHASING - USED CARS
1110	1765	D04	PURCHASING - NEW CARS
1120	2004	D06	PURCHASING - SERVICE
2200	2180	D02	SALES - USED CARS
2210	2010	D04	SALES - NEW CARS
3510	3082	D02	APPRAISAL - USED CARS
3520	3769	D04	APPRAISAL NEW CARS
3530	2209	D06	APPRAISAL - SERVICE
4040	1234	D09	Audit
4200	1003	D04	LEASING - NEW CARS
4500	3222	D09	HUMAN RESOURCES
4600	2096	D06	MAINTENANCE
4900	2466	D09	MIS
5000	2466	D09	CORPORATE ACCOUNTING
5050	<null>	D09	RESEARCH
5100	2598	D06	BILLING
5200	2894	D09	CORPORATE MARKETING
6000	1003	D09	LEGAL
6060	<null>	D09	Claims
6200	2461	D09	CORPORATE ADMINISTRATION

20 rows processed

8.2.2 Exercise 8-2

Try a few more: Add two more departments that you choose. Leave the department head column null.

Enter a SELECT statement to display the table to confirm the insertions. To check your answers, see “Exercise 8-2 answer” on page B-27.

8.2.3 Exercise 8-3

Try another: A project is about to get underway. You know that the project ID is P434 and that its name is Mass Media Campaign Blitz.

Insert a row into the PROJECT table giving this information. The PROJ_ID and PROJ_DESC columns contain character data.

Confirm the insertion. To check your answers, see “Exercise 8-3 answer” on page B-27. The result looks like this:

PROJ_ID	PROJ_DESC
C200	New brand research
C203	Consumer study
C240	Service study
P400	Christmas media
P434	Mass Media Campaign Blitz
P634	TV ads - WTVK
D880	Systems analysis

7 rows processed

Inserting rows with SELECT: You can copy selected rows from one table and put them into another table using the SELECT statement within the INSERT statement. A SELECT statement in an INSERT statement is referred to as a **query specification**.

Include a WHERE clause in the SELECT statement to insert only those rows meeting the search condition.

How it's done: Your company has decided to hire one of the consultants, 9388, as a full-time employee. To add the employee information from the CONSULTANT table into the ASSIGNMENT table, enter:

```
insert into assignment (emp_id, proj_id, start_date)
      select con_id, proj_id, start_date
      from consultant
      where con_id = 9388;
```

The columns in the receiving table must have data compatible with the data of the corresponding columns in the sending table.

You can use SELECT * if you are using all columns in the same order as in the table to which you are adding the row.

Enter a SELECT statement to display the table and confirm the insertion.

The result looks like this:

EMP_ID	PROJ_ID	START_DATE	END_DATE
2894	P634	2000-02-15	<null>
3411	P634	2000-03-01	<null>
4358	C240	1998-06-01	1998-08-15
2466	D880	1999-11-01	<null>
9388	D880	1997-12-21	<null>

5 rows processed

8.3 Modifying data in a table with SET

You use the UPDATE statement to modify columns or rows in a table.

Modifying values in a column: If you want to modify every value in a column throughout the table, you need to:

- Identify the table you intend to modify by specifying UPDATE and the table name
- Name the column in which the modification is to take place and give the new data with a SET clause

How it's done: At certain times during the year, every employee's accrued vacation is increased by eight hours.

8.3.1 Exercise 8-4

Enter a SELECT statement to display the employees and vacation hours accrued. To check your SELECT statement, see “Exercise 8-4 answer” on page B-27. Jot down a couple of the employees and their vacation hours accrued so you can check them after you have made the changes.

To make the vacation hour changes, enter:

```
update benefits
  set vac_accrued = vac_accrued + 8
  where fiscal_year = 2000;
```

What message do you see?

You see a message specifying the number of rows that have been updated.

Display the changes: Enter a SELECT statement sorted by employee id to display the updated table.

The result looks like this:

```
EMP_ID  VAC_ACCRUED
-----  -
1003    100.00
1034    100.50
1234    100.00
1765    100.50
2004    100.50
2010    100.75
2096    100.50
2106    100.50
2174    100.00
2180    100.50
2209    100.50
2246    100.50
2424    100.50
2437    76.00
2448    76.00
2461    76.00
2466    100.50
2598    68.00
2781    76.00
2894    76.00
3082    76.00
3118    76.00
3222    76.00
3288    76.00
3294    76.00
3338    76.00
3341    76.00
3411    76.00
3433    76.00
3449    76.00
3704    76.00
3764    76.00
3767    76.00
3769    76.00
3778    76.00
3841    76.00
3991    76.00
4001    76.00
4002    76.00
4008    76.00
4027    76.00
4321    76.00
4358    76.00
4456    76.00
4660    76.00
4703    54.75
4773    76.00
4962    76.00
5008    54.50
5090    54.00
5103    54.00
```

51 rows processed

Modifying selected rows: Often you want to change the value in a column only in rows that meet a certain search condition.

How it's done: All employees who have accrued more than 80 hours of vacation time are supposed to have an additional eight hours added to their accrued vacation. To do this, add a **WHERE** clause to the previous statement:

```
update benefits
  set vac_accrued = vac_accrued + 8
  where vac_accrued > 80
  and fiscal_year = 2000;
```

Display the changes: Enter a SELECT statement to display the BENEFITS table to confirm the change.

The result looks like this:

EMP_ID	VAC_ACCRUED
2010	108.75
2246	108.50
1034	108.50
2424	108.50
2209	108.50
1765	108.50
2180	108.50
2106	108.50
2096	108.50
2004	108.50
1003	108.00
1234	108.00
2466	108.50
2174	108.00

14 rows processed

The only rows for which that column is updated are the rows that meet the condition.

8.3.2 Exercise 8-5

Now you try it: It was recently discovered that the name of department 6060 is incorrect. Instead of Claims, it should be Lost Claims. Make the appropriate change to the DEPARTMENT table.

Confirm the modification by writing an appropriate SELECT statement. To check your answers, see “Exercise 8-5 answer” on page B-27.

8.3.3 Exercise 8-6

Try another: Update the EMPLOYEE table so that employee 3433 is associated with department 6200.

Confirm the modification by issuing an appropriate SELECT statement. To check your answers, see “Exercise 8-6 answer” on page B-27.

8.3.4 Exercise 8-7

And another: Three employees (1034, 3704, and 4660) have recently moved to Framingham. Update the EMPLOYEE table appropriately.

Confirm the modification by issuing an appropriate SELECT statement. To check your answers, see “Exercise 8-7 answer” on page B-27.

8.4 Removing data from a table

Use the DELETE statement to remove one or more rows from a database table.

How it's done: One of the departments you entered previously, department 4040, shouldn't exist at all. Delete it by entering:

```
delete from department
  where dept_id = 4040;
```

This will delete only information on department 4040.

What message do you see?

You see a message stating that one row was deleted.

Display the changes: Enter a SELECT statement to display the DEPARTMENT table to confirm the deletion.

The result looks like this:

DEPT_ID	DEPT_HEAD_ID	DIV_CODE	DEPT_NAME
1100	2246	D02	PURCHASING - USED CARS
1110	1765	D04	PURCHASING - NEW CARS
1120	2004	D06	PURCHASING - SERVICE
2200	2180	D02	SALES - USED CARS
2210	2010	D04	SALES - NEW CARS
3510	3082	D02	APPRAISAL - USED CARS
3520	3769	D04	APPRAISAL NEW CARS
3530	2209	D06	APPRAISAL - SERVICE
4200	1003	D04	LEASING - NEW CARS
4500	3222	D09	HUMAN RESOURCES
4600	2096	D06	MAINTENANCE
4900	2466	D09	MIS
5000	2466	D09	CORPORATE ACCOUNTING
5100	2598	D06	BILLING
5200	2894	D09	CORPORATE MARKETING
6000	1003	D09	LEGAL
6060	<null>	D09	Claims
6200	2461	D09	CORPORATE ADMINISTRATION

19 rows processed

8.4.1 Exercise 8-8

Now you try it: Department 5050 also should not be in the database. Delete it from the DEPARTMENT table.

Enter a SELECT statement to confirm the deletion. To check your answers, see “Exercise 8-8 answer” on page B-28.

Omitting the WHERE clause: If you do not use the WHERE clause in the DELETE statement, *all* rows of data are deleted from the table. The empty table remains in the database.

8.4.2 Exercise 8-9

Try another: Create a DELETE statement to erase department 6060 and the other two departments you added from the DEPARTMENT table while practicing INSERT.

Confirm the deletion by issuing an appropriate SELECT statement. To check your answers, see “Exercise 8-9 answer” on page B-28.

8.5 Review

Choose the correct answers for the questions below. More than one answer can apply for each question.

1. You use a `SELECT` statement with `INSERT` to:
 - a. Copy specific rows from one table to another
 - b. Add a completely new row to a database
 - c. Modify a row in a table
2. If you don't have a value for every column you are adding to a table, you can:
 - a. Identify only the columns you are going to insert values into
 - b. Use the keyword `NULL` for the columns where the value is unknown
 - c. Use two quotes with a space between for the columns where the value is unknown
3. You can update all rows in a table by:
 - a. Using a `WHERE` clause with an `*`
 - b. Omitting the `WHERE` clause
 - c. Using an `*` after the keyword `UPDATE`
4. You can update selected rows in a table by:
 - a. Specifying columns after the keyword `UPDATE`
 - b. Using a `WHERE` clause with an `*`
 - c. Specifying a search condition in a `WHERE` clause
5. You are updating all columns in a table but do not know the specific value to put into one column. You can:
 - a. Use the keyword `UNKNOWN` for the column where the value is unknown
 - b. Use two quotes with a space between for the column where the value is unknown
 - c. Use the keyword `NULL` for the column where the value is unknown
6. If you do not have a `WHERE` clause in a `DELETE` statement:
 - a. All the rows are deleted and the table is deleted as well
 - b. The table is deleted
 - c. All the rows are deleted but the table remains

To check your answers, see "Review answers" on page B-28.

Part III. Appendixes

Appendix A. Sample Data Description Language

- A.1 About this appendix A-3
- A.2 Table creation A-4
- A.3 Indexes A-6
- A.4 Views A-7
- A.5 Data integrity A-8

A.1 About this appendix

Up to now, you have been using SQL DML statements to retrieve or update data. You haven't been concerned with table design or layout, and you have had full access to all tables. This appendix shows how a system administrator might use DDL to define a database and users' access to it. If you use SQL against a multi-user database, your use can be affected by decisions made by the system administrator who:

- Creates databases and tables using SQL DDL
- Specifies which database the tables are associated with and identifies the physical files in which to store data
- Defines constraints on the data so that only valid data is stored in the database
- Restricts access to tables and creates views of them

Each of these functions can have an impact on the way you access data from tables.

No online exercises: There are no online exercises in this chapter. Please *do not* enter any of the statements online.

A.2 Table creation

Before you can access a table, it must be defined to the database management system.

For example, to set up the EMPLOYEE table, you or the system administrator uses an appropriate tool to create the following DDL statement:

```
create table employee
(emp_id          integer      not null,
 manager_id     integer
 emp_fname      varchar(20)  not null,
 emp_lname      varchar(20)  not null,
 dept_id        integer      not null,
 street         varchar(40)  not null,
 city           char(20)     not null,
 state          char(04)     not null,
 zip_code       char(09)     not null,
 phone          char(10),
 status         char(01),
 ss_number      integer      not null,
 start_date     date         not null,
 termination_date date,
 birth_date     date         not null);
```

The system administrator specifies the column names and the kind of data you can put into each column.

Common columns: The system administrator identifies potential relationships between tables and plans for common columns that can become foreign keys. You use these common columns when you want to join tables.

Temporary tables: In a multi-user database environment, there are times when you need to store data for only a short period of time, perhaps as long as a program is running. When tables are set up for this purpose, they are called **temporary tables**.

The data in temporary tables is not stored in the database and is not accessible to other users.

For example, you may need to access all the company's accounts receivable data for the past five years. This information is stored in an accounts receivable history table that covers the past 15 years. You want to retrieve data in different forms, so you'll use several SELECT statements.

Rather than access the very large history table every time you need to retrieve data, you can define a temporary table that has data from just the past five years and retrieve from this smaller table. This allows you quicker and more efficient access to the data.

A.3 Indexes

An index is a way to order a table logically to speed the retrieval of data.

For example, an index could be defined to order the EMPLOYEE table by employee last name. Another index could be established to order the same table by social security number.

An index is established on a column or combination of columns:

- To improve processing efficiency
- To prevent duplicate rows (when an index is specified as unique)

Identifying an index: The system administrator examines all the columns and the programs that will run against the table to determine how the data is most likely to be accessed.

For example, assume several programs access the EMPLOYEE table to list employees alphabetically by last name. The system administrator would place an index on the employee last name to allow these programs to access the data efficiently.

There can be several indexes associated with one table.

When you issue a SELECT statement, SQL uses the indexes to access the data as efficiently as possible.

Creating an index: To create an index on employee last name, the system administrator uses the following DDL statement:

```
create index lname_index
  on employee (emp_lname);
```

Indexes can be added or dropped as necessary.

A.4 Views

You don't always need to see an entire table. In fact, a table may contain data (like salary information) that shouldn't be seen by everyone.

You may also frequently want to see two or more tables together.

In these situations, the systems administrator creates a view.

What is a view: A view is defined using DDL and can be:

- A subset of columns and rows in one or more tables
- Two or more joined tables

A view is represented internally by a stored command, not stored data. A view can display data from one or more tables or from other views.

Each view has a name, just as a table has a name. You can use all SQL SELECT commands that you use against a table against a view as well.

When you display the view, you see only the *subset* of columns and rows specified in the view definition.

Here is a table and one view of it:

POSITION

EMP_ID	JOB_ID	SALARY_AMOUNT
2096	4666	42640
2437	4560	21944
2598	2053	13520

POS_INFO

EMP_ID	JOB_ID
2096	4666
2437	4560
2598	2053

A.5 Data integrity

Data integrity means correctness of data throughout the database. When you add data to a table, you want that data to be correct and consistent with other data in the table.

If you add an employee to the EMPLOYEE table, you want to make sure that the employee has an ID. Every employee must have an ID to keep the employee entries consistent with one another. This is an example of data integrity.

In other cases, the value in a given column cannot be repeated. The employee ID is unique for each employee; two employees cannot have the same ID. This is another example of data integrity.

To ensure that the data is consistent across tables, integrity constraints are set up as part of the data definition. The data you enter is checked against these constraints.

Appendix B. Answers to Exercises

B.1 Chapter 1	B-3
B.2 Chapter 2	B-4
B.3 Chapter 3	B-5
B.4 Chapter 4	B-10
B.5 Chapter 5	B-14
B.6 Chapter 6	B-19
B.7 Chapter 7	B-24
B.8 Chapter 8	B-27

B.1 Chapter 1

Review answers: These are the answers for 1.6, “Review” on page 1-11.

Description	Term
1. Components of a relational database that hold the data	c. Tables
2. Components of a table	d. Rows and columns
3. A column or combination of columns holding values that form the primary key of another table	a. Foreign key e. Select, project, and join
4. The types of operations you can perform against a relational database	a. Foreign key
5. A way to establish a relationship between two tables	b. Primary key
6. A column or combination of columns that uniquely identifies a row in a table	

B.2 Chapter 2

Review answers: These are the answers for 2.5, “Review” on page 2-8.

1. SQL stands for **Structured Query Language**.
2. You use SQL **data description language (DDL)** statements to define tables.
3. You use SQL **data manipulation language (DML)** statements to change data in a table.
4. The three SQL update operations are **INSERT, UPDATE, and DELETE**.
5. An interactive SQL statement ends with a **delimiter/semicolon**.
6. An interactive SQL statement begins with a **verb**.
7. An interactive SQL statement **can** span several lines.
8. You **cannot** issue several interactive SQL statements at once.
9. Interactive SQL gives you **immediate** results.
10. Embedded SQL returns the results to the **program**.

B.3 Chapter 3

Exercise 3-1 answer: This is the answer for 3.2.1, “Exercise 3-1” on page 3-5.

```
select *  
  from skill;
```

Exercise 3-2 answer: This is the answer for 3.3.1, “Exercise 3-2” on page 3-8.

```
select skill_id, skill_name  
  from skill;
```

Exercise 3-3 answer: This is the answer for 3.3.2, “Exercise 3-3” on page 3-8.

```
select emp_fname, emp_lname, street, city  
  from employee;
```

Exercise 3-4 answer: This is the answer for 3.4.1, “Exercise 3-4” on page 3-11.

```
select dept_id as "Department ID", dept_name as "Name"  
  from department;
```

Exercise 3-5 answer: This is the answer for 3.5.1, “Exercise 3-5” on page 3-13.

```
select emp_id as "Employee",  
       salary_amount as "Salary",  
       bonus_percent as "Bonus Percentage",  
       bonus_percent * salary_amount as "Bonus Paid"  
  from position;
```

Exercise 3-6 answer: This is the answer for 3.6.1, “Exercise 3-6” on page 3-26.

```
select distinct city  
  from employee;
```

Exercise 3-7 answer: This is the answer for 3.7.1, “Exercise 3-7” on page 3-30.

```
select emp_id, emp_lname  
  from employee  
  order by emp_lname desc;
```

Exercise 3-8 answer: This is the answer for 3.7.2, “Exercise 3-8” on page 3-31.

```
select skill_id, skill_name  
  from skill  
  order by skill_id;
```

Exercise 3-9 answer: This is the answer for 3.7.3, “Exercise 3-9” on page 3-33.

```
select dept_id, emp_lname, emp_id  
  from employee  
  order by dept_id, emp_lname;
```

Exercise 3-10 answer: This is the answer for 3.7.4, “Exercise 3-10” on page 3-36.

```
select emp_id as "Employee",
       salary_amount as "Base Salary",
       bonus_percent as "Bonus Percentage",
       bonus_percent * salary_amount as "Bonus Paid"
from position
order by 4;
```

Review answers: These are the answers for 3.8, “Review” on page 3-38.

1. How many columns would be retrieved by the following statement:

```
select * from sample_table;
```

c. All

2. How can you limit the number of columns returned by your SELECT statement?

b. List the columns you want to see

3. How can you give heading names to columns?

b. Use AS after the column name and specify the heading

4. Given a table called SUPPLY_PRICE and a column in that table called PART_NUMBER, which of the following statements will find the number of *unique* part numbers in the table?

c. select distinct part_number from supply_price;

5. How can you name the column you want to sort by?

a. Use the column name

b. Use the heading name

d. Use the column number

Scenario answers: These are the answers for 3.9, “Scenarios” on page 3-39.

1. You need to list all jobs the company has for a government screen. The screen should show job ID, job title, and minimum and maximum rate for the job. Use the JOB table, checking Appendix C for table descriptions.

```
select job_id, job_title, min_rate, max_rate
from job;
```

JOB_ID	JOB_TITLE	MIN_RATE	MAX_RATE
8001	Vice President	90000.00	136000.00
2077	Purch Clerk	17000.00	30000.00
9001	President	111000.00	190000.00
3051	Data Entry Clerk	8.50	11.45
4700	Purch Agnt	33000.00	60000.00
3029	Computer Operator	25000.00	44000.00
6011	Manager - Acctng	59400.00	121000.00
4130	Benefits Analyst	35000.00	56000.00
4666	Sr Mechanic	41000.00	91000.00
4123	Recruiter	35000.00	56000.00
5555	Salesperson	30000.00	79000.00
4025	Writer - Mktng	31000.00	50000.00
4023	Accountant	44000.00	120000.00
2051	AP Clerk	8.80	14.60
4734	Mktng Admin	25000.00	62000.00
5110	CUST SER MGR	40000.00	108000.00
2053	AR Clerk	8.80	14.60
6004	Manager - HR	66000.00	138000.00
5111	CUST SER REP	27000.00	54000.00
4012	Admin Asst	21000.00	44000.00
2055	PAYROLL CLERK	17000.00	30000.00
4560	Mechanic	11.45	21.00
5890	Appraisal Spec	45000.00	70000.00
3333	Sales Trainee	21600.00	39000.00
6021	Manager - Mktng	76000.00	150000.00

25 rows processed

2. The screen you just created (in Scenario 1) has all the necessary information but is difficult to read because it is not sorted. Modify the SELECT statement to create the same screen sorted by job title.

```
select job_id, job_title, min_rate, max_rate
       from job
       order by job_title;
```

JOB_ID	JOB_TITLE	MIN_RATE	MAX_RATE
4023	Accountant	44000.00	120000.00
4012	Admin Asst	21000.00	44000.00
5890	Appraisal Spec	45000.00	70000.00
2051	AP Clerk	8.80	14.60
2053	AR Clerk	8.80	14.60
4130	Benefits Analyst	35000.00	56000.00
3029	Computer Operator	25000.00	44000.00
5110	CUST SER MGR	40000.00	108000.00
5111	CUST SER REP	27000.00	54000.00
3051	Data Entry Clerk	8.50	11.45
6011	Manager - Acctng	59400.00	121000.00
6004	Manager - HR	66000.00	138000.00
6021	Manager - Mktng	76000.00	150000.00
4560	Mechanic	11.45	21.00
4734	Mktng Admin	25000.00	62000.00
9001	President	111000.00	190000.00
4700	Purch Agnt	33000.00	60000.00
2077	Purch Clerk	17000.00	30000.00
2055	PAYROLL CLERK	17000.00	30000.00
4123	Recruiter	35000.00	56000.00
3333	Sales Trainee	21600.00	39000.00
5555	Salesperson	30000.00	79000.00
4666	Sr Mechanic	41000.00	91000.00
8001	Vice President	90000.00	136000.00
4025	Writer - Mktng	31000.00	50000.00

25 rows processed

3. Periodically, a company list is produced showing each department and all employees assigned to that department. This list should be sorted first by department ID and then by employee ID within each department. Display department ID, employee ID, and employee last name. Create the appropriate SQL SELECT statement to produce this list using the EMPLOYEE table.

```
select dept_id, emp_id, emp_lname
       from employee
       order by dept_id, emp_id;
```

DEPT_ID	EMP_ID	EMP_LNAME
1100	2246	Hamel
1100	4703	Halloran
1100	5008	Fordman
1110	1765	Alexander
1110	2106	Widman
1120	2004	Johnson
1120	2898	Umidy
1120	3294	Johnson
1120	3338	White
2200	2180	Albertini
2200	2448	Lynn
2200	3704	Moore
2200	3767	Lowe
2200	4660	MacGregor
	.	
	.	
	.	

55 rows processed

4. The screen you just created is very useful except that the headings are difficult to understand. Rewrite the statement so that the column names are "Department", "Employee ID", and "Last Name".

```
select dept_id as "Department", emp_id as "Employee ID",  
       emp_lname as "Last Name"  
from employee  
order by 1, 2;
```

Department	Employee ID	Last Name
1100	2246	Hamel
1100	4703	Halloran
1100	5008	Fordman
1110	1765	Alexander
1110	2106	Widman
1120	2004	Johnson
1120	2898	Umidy
1120	3294	Johnson
1120	3338	White
2200	2180	Albertini
2200	2448	Lynn
2200	3704	Moore
2200	3767	Lowe
2200	4660	MacGregor
.	.	.
.	.	.
.	.	.

55 rows processed

B.4 Chapter 4

Exercise 4-1 answer: This is the answer for 4.4.1, “Exercise 4-1” on page 4-7.

```
select emp_id, emp_fname, emp_lname
       from employee
       where emp_id = 5103;
```

Exercise 4-2 answer: This is the answer for 4.4.2, “Exercise 4-2” on page 4-8.

```
select emp_id, job_id, salary_amount
       from position
       where salary_amount > 100000;
```

Exercise 4-3 answer: This is the answer for 4.4.3, “Exercise 4-3” on page 4-9.

```
select emp_id, emp_fname, emp_lname, city
       from employee
       where city = 'Boston'
       order by emp_id;
```

Exercise 4-4 answer: This is the answer for 4.4.4, “Exercise 4-4” on page 4-9.

```
select emp_id, emp_fname, emp_lname, city
       from employee
       where not city = 'Boston';
```

Exercise 4-5 answer: This is the answer for 4.5.1, “Exercise 4-5” on page 4-11.

```
select emp_id
       from position
       where bonus_percent is null;
```

Exercise 4-6 answer: This is the answer for 4.5.2, “Exercise 4-6” on page 4-13.

```
select emp_id, emp_fname, emp_lname, phone
       from employee
       where phone is not null;
```

Exercise 4-7 answer: This is the answer for 4.5.3, “Exercise 4-7” on page 4-15.

```
select job_id, emp_id, salary_amount
       from position
       where salary_amount between 20000 and 35000;
```

Exercise 4-8 answer: This is the answer for 4.5.4, “Exercise 4-8” on page 4-17.

```
select emp_id, salary_amount
       from position
       where salary_amount in (41600, 45240, 50440);
```

Exercise 4-9 answer: This is the answer for 4.5.5, “Exercise 4-9” on page 4-19.

```
select dept_id, dept_name from department
       where dept_name like '%NEW CARS%';
```

Exercise 4-10 answer: This is the answer for 4.5.6, “Exercise 4-10” on page 4-19.

```
select dept_id, dept_name from department
       where dept_name not like '%NEW CARS%';
```

Exercise 4-11 answer: This is the answer for 4.6.1, “Exercise 4-11” on page 4-21.

```
select proj_id, est_man_hours - act_man_hours
       from project
       where est_man_hours - act_man_hours > 0;
```

Exercise 4-12 answer: This is the answer for 4.7.1, “Exercise 4-12” on page 4-22.

```
select emp_id, city, phone
       from employee
       where city in ('Camden', 'Brookline', 'Canton')
          or phone is not null;
```

Review answers: These are the answers for 4.8, “Review” on page 4-27.

1. The clause that allows the user to specify search conditions that filter the rows to be selected is:
 - a. **The WHERE clause**
2. What are the components of the WHERE clause?
 - b. **The keyword WHERE**
 - c. **A search condition**
3. You can compare a character column to a
 - b. **Mask**
4. Masks are used with
 - c. **The LIKE predicate**
5. Which of the following are mask characters?
 - b. **_**
 - d. **%**
6. The IS NULL predicate causes:
 - b. **The retrieval of rows where a column contains no value**
7. Parentheses are used to:
 - a. **Set up the sequence of arithmetic evaluation**
 - b. **Set up the sequence of evaluation of AND and OR in a compound WHERE clause**

Scenario answers: These are the answers for 4.9, “Scenarios” on page 4-29.

1. Periodically, a list is published giving divisions and their departments. A new department was recently added to division D09, so a new list for that division is needed. Use the DEPARTMENT table and show division code, department ID, and department name. Order by department ID.

```
select div_code, dept_id, dept_name
       from department
       where div_code = 'D09'
       order by dept_id;
```

DIV_CODE	DEPT_ID	DEPT_NAME
D09	4500	HUMAN RESOURCES
D09	4900	MIS
D09	5000	CORPORATE ACCOUNTING
D09	5200	CORPORATE MARKETING
D09	6000	LEGAL
D09	6200	CORPORATE ADMINISTRATION

6 rows processed

2. All Commonwealth Auto employees whose last names begin with L and M are due to have flu shots. The medical office needs to have the complete names of these individuals and the department to which each is assigned. Sort the list by last name. (Use the EMPLOYEE table.)

```
select emp_lname, emp_fname, dept_id
       from employee
       where emp_lname like 'L%' or emp_lname like 'M%'
       order by emp_lname;
```

EMP_LNAME	EMP_FNAME	DEPT_ID
Loren	Martin	4600
Lowe	Frank	2200
Lynn	David	2200
MacGregor	Bruce	2200
Mills	Thomas	6200
Moore	Richard	2200

6 rows processed

3. The Marketing department has a large project coming up and needs employees who have at least a medium level of competence (greater than 02) in skill 3333. Display employee ID and level of competence for each employee using the EXPERTISE table.

```
select emp_id, skill_level
       from expertise
       where skill_id = 3333
          and skill_level > '02';
```

```

EMP_ID  SKILL_LEVEL
-----  -
2437   04
3288   04

2 rows processed

```

4. In order to identify employees involved in media projects, the Human Resources department needs a list of employees associated with a project ID that begins with P (indicating media-related). Order the list by employee ID. (Use the ASSIGNMENT table to find this information.)

```

select emp_id, proj_id
       from assignment
       where proj_id like 'P%'
       order by emp_id;

```

```

EMP_ID  PROJ_ID
-----  -
2894   P634
3411   P634

2 rows processed

```

5. The budget group needs a list of employees who hold a position that pays less than \$25,000. Show employee ID and salary.

```

select emp_id, salary_amount
       from position
       where salary_amount < 25000;

```

```

EMP_ID  SALARY_AMOUNT
-----  -
3338    22048.84
3767    2200.00
4660    24000.00
1765    18001.00
2180    19000.10
2106    23920.00
3704    22880.00
4008    24441.00
4703    24857.00

9 rows processed

```

B.5 Chapter 5

Exercise 5-1 answer: This is the answer for 5.2.1, “Exercise 5-1” on page 5-5.

```
select avg(num_dependents)
       from coverage;
```

Exercise 5-2 answer: This is the answer for 5.2.2, “Exercise 5-2” on page 5-5.

```
select avg(vac_accrued)
       from benefits
       where fiscal_year = 1999;
```

Exercise 5-3 answer: This is the answer for 5.2.3, “Exercise 5-3” on page 5-6.

```
select count(*)
       from skill;
```

Exercise 5-4 answer: This is the answer for 5.2.4, “Exercise 5-4” on page 5-7.

```
select count(phone)
       from employee
       where dept_id = 5200;
```

Exercise 5-5 answer: This is the answer for 5.2.5, “Exercise 5-5” on page 5-8.

```
select max(salary_amount)
       from position
       where job_id = 3333;
```

Exercise 5-6 answer: This is the answer for 5.2.6, “Exercise 5-6” on page 5-8.

```
select min(sick_taken)
       from benefits;
```

Exercise 5-7 answer: This is the answer for 5.2.7, “Exercise 5-7” on page 5-9.

```
select sum(vac_taken)
       from benefits;
```

Exercise 5-8 answer: This is the answer for 5.3.1, “Exercise 5-8” on page 5-10.

```
select count(distinct proj_id)
       from consultant;
```

Exercise 5-9 answer: This is the answer for 5.4.1, “Exercise 5-9” on page 5-12.

```
select dept_id, count(emp_id)
       from employee
       group by dept_id;
```

Exercise 5-10 answer: This is the answer for 5.4.2, “Exercise 5-10” on page 5-12.

```
select job_id, sum(salary_amount)
       from position
       group by job_id;
```

Exercise 5-11 answer: This is the answer for 5.5.1, “Exercise 5-11” on page 5-15.

```
select city, count(emp_id)
       from employee
       group by city
       having count(emp_id) > 2;
```

Exercise 5-12 answer: This is the answer for 5.5.2, “Exercise 5-12” on page 5-16.

```
select job_id, avg(salary_amount)
       from position
       group by job_id
       having avg(salary_amount) > 25000;
```

Exercise 5-13 answer: This is the answer for 5.6.1, “Exercise 5-13” on page 5-17.

```
select job_id, avg(salary_amount) as "Average Salary"
       from position
       group by job_id
       having avg(salary_amount) > 25000;
```

Review answers: These are the answers for 5.7, “Review” on page 5-18.

1. You use **aggregate functions** to perform calculations within a **SELECT** statement.
2. An aggregate function can be used instead of a column name with **SELECT** or in the **HAVING** clause.
3. When the aggregate function **AVG** encounters a null value, it **ignores** the row.
4. The **HAVING** clause acts as a search condition with an aggregate function.
5. You rename an aggregate function column heading by using **AS** and the heading you want.

Scenario answers: These are the answers for 5.8, “Scenarios” on page 5-19.

1. In order to plan for the Christmas party for Commonwealth Auto, the Human Resources department needs a count of employees by department. (The **EMPLOYEE** table contains this information.)

```
select dept_id, count(emp_id)
       from employee
       group by dept_id;
```

DEPT_ID	(EXPR)
1100	3
1110	2
1120	4
2200	5
2210	8
3510	2
3520	1
3530	2
4500	3
4600	9
5000	3
5100	2
5200	5
6200	6

14 rows processed

2. As part of its salary research, the Human Resources department needs to know the minimum and maximum salaries being earned for each job ID in the company. (Use the POSITION table.)

```
select job_id, min(salary_amount), max(salary_amount)
       from position
       group by job_id;
```

JOB_ID	(EXPR)	(EXPR)
-----	-----	-----
2051	<null>	<null>
2053	<null>	<null>
2077	18001.00	29536.00
3333	2200.00	30680.00
4012	28601.80	44001.40
4023	74776.00	74776.00
4025	43888.00	43888.00
4123	49921.76	49921.76
4130	45241.94	45241.94
4560	<null>	<null>
4666	85280.00	85280.00
4700	47009.34	59488.00
4734	53665.00	57824.50
5110	56977.80	56977.80
5555	36400.00	76961.00
5890	41600.00	68016.00
6004	110448.00	110448.00
6011	94953.52	94953.52
6021	111593.00	111593.00
8001	117832.68	117832.68
9001	146432.00	146432.00

21 rows processed

3. Upper management needs to know how many subordinate employees there are for each manager in order to evaluate the span of control within the company. The EMPLOYEE table contains this information.

```
select manager_id, count(emp_id)
       from employee
       group by manager_id;
```

MANAGER_ID	(EXPR)
1003	5
1034	3
1234	1
1765	1
2004	2
2010	6
2096	3
2180	3
2209	1
2246	2
2448	1
2461	2
2466	6
2894	7
3082	1
3222	2
3778	1
3991	1
4321	1
4358	1
<null>	5

21 rows processed

4. A project is coming up that requires project members having the skill ID 3333 (body work). The project leader needs to find out how many employees have a skill level greater than 02 for this skill to see whether he needs to hire consultants to staff the project. Keep in mind that the SKILL_LEVEL column contains character data. (Use the EXPERTISE table.)

```
select count(emp_id)
  from expertise
 where skill_id = 3333
    and skill_level > '02';
```

(EXPR)
2

1 row processed

5. The Human Resources department is conducting research into salaries. They have asked you for a screen showing:

- Job ID
- Average salary by job
- Minimum salary by job
- Maximum salary by job

They need this screen only for current positions with a job ID less than 4000 (indicating training and clerical positions) where the average salary is less than \$25,000. Use the POSITION table and rename the column headings so that the screen makes sense.

```

select job_id as "Job",
       avg(salary_amount) as "Average Salary",
       min(salary_amount) as "Minimum Salary",
       max(salary_amount) as "Maximum Salary"
  from position
  where job_id < 4000
  group by job_id
  having avg(salary_amount) < 25000;

```

Job	Average Salary	Minimum Salary	Maximum Salary
2077	23672.56	18001.00	29536.00
3333	23130.05	2200.00	30680.00

2 rows processed

6. The training group is concerned that there are few people in the company who have certain crucial skills. They have asked you to give them a screen listing the number of employees who have either a medium level of competence (02 or above) for skill 3333 (body work) or a high level of competence (04) for skill 4444 (assembly). The screen should list a skill only if there are more than two employees that fit that category.

```

select skill_id, count(emp_id)
  from expertise
  where (skill_id = 3333 and skill_level >= '02')
     or (skill_id = 4444 and skill_level = '04')
  group by skill_id
  having count(emp_id) > 2;

```

SKILL_ID	(EXPR)
3333	3

1 row processed

B.6 Chapter 6

Exercise 6-1 answer: This is the answer for 6.2.1.1, “Exercise 6-1” on page 6-6.

```
select div_code, div_name, emp_id, emp_lname, emp_fname
       from employee, division
       where emp_id=div_head_id;
```

Exercise 6-2 answer: This is the answer for 6.2.3.1, “Exercise 6-2” on page 6-9.

```
select expertise.emp_id, emp_lname,
       emp_fname, skill_id
       from demoproj.expertise, demoempl.employee
       where expertise.emp_id = employee.emp_id;
```

Exercise 6-3 answer: This is the answer for 6.2.4.1, “Exercise 6-3” on page 6-10.

```
select expertise.emp_id, emp_lname, emp_fname, skill_id
       from demoproj.expertise, demoempl.employee
       where employee.emp_id = expertise.emp_id
       order by emp_lname desc, emp_fname desc;
```

Exercise 6-4 answer: This is the answer for 6.2.6, “Exercise 6-4” on page 6-12.

```
select distinct consultant.manager_id, emp_lname, emp_fname
       from demoproj.consultant, demoempl.employee
       where consultant.manager_id = employee.emp_id;
```

Review answers: These are the answers for 6.5, “Review” on page 6-18.

Statement	Term
1. Needed to join two or more tables	b. Common columns
2. Resolves the problem of joining table columns that have the same name	f. Aliases
3. Where the joining is specified	a. The WHERE clause
4. Where an alias is identified	d. The FROM clause
5. Used to append one table to another	c. The UNION clause

Scenario answers: These are the answers for 6.6, “Scenarios” on page 6-19.

1. Management would like to see which employees are involved in which projects. Write a `SELECT` statement to retrieve this information by joining the `ASSIGNMENT` and `PROJECT` tables that contain the data. Display the information by project description.

```
select assignment.proj_id, proj_desc, emp_id
       from assignment, project
       where project.proj_id = assignment.proj_id
       order by proj_desc;
```

PROJ_ID	PROJ_DESC	EMP_ID
C203	Consumer study	2894
C240	Service study	4358
D880	Systems analysis	2466
D880	Systems analysis	9388
P634	TV ads - WTVK	3411

5 rows processed

2. The Human Resources department needs a list of employees and their remaining vacation time. This information is contained in the `EMPLOYEE` and `BENEFITS` tables. Display employee ID and last name as well as the vacation time remaining in fiscal year 2000. Order your screen by employee ID.

```
select benefits.emp_id, emp_lname,
       (vac_accrued - vac_taken)
       from benefits, employee
       where benefits.emp_id = employee.emp_id
             and fiscal_year = 2000
       order by benefits.emp_id;
```

EMP_ID	EMP_LNAME	(EXPR)
1003	Baldwin	108.00
1034	Galloway	36.50
1234	Mills	68.00
1765	Alexander	76.50
2004	Johnson	68.50
2010	Parker	92.75
2096	Carlson	28.50
2106	Widman	76.50
2174	Zander	60.00
2180	Albertini	108.50
2209	Smith	76.50
2246	Hamel	36.50
2424	Wilder	60.50
2437	Thompson	76.00
2448	Lynn	55.50
2461	Anderson	36.00
2466	Bennett	68.50
2598	Jacobs	60.00
2781	Thurston	16.00
2894	Griffin	76.00
3082	Brooks	24.00
3118	Wooding	68.00
3222	Voltmer	76.00
3288	Sampson	20.00
3294	Johnson	60.00
3338	White	76.00
3341	Smith	43.50
3411	Williams	8.00
3433	Crane	36.00
3449	Taylor	20.00
3704	Moore	28.00
3764	Park	-4.00
3767	Lowe	8.00
3769	Donelson	76.00
3778	Ferndale	36.00
3841	Cromwell	76.00
3991	Wilkins	8.00
4001	Thompson	36.00
4002	Roy	36.00
4008	Clark	76.00
4027	Courtney	36.00
4321	Bradley	28.00
4358	Robinson	76.00
4456	Thompson	36.00
4660	MacGregor	20.00
4703	Halloran	38.75
4773	Dexter	8.00
4962	White	60.00
5008	Fordman	14.50
5090	Wills	54.00
5103	Ferguson	54.00

51 rows processed

3. More statistics are being gathered on vacation hours. You have been asked to produce a screen of average vacation hours taken for each department. Display department ID and average vacation taken for fiscal 1999. Order the screen by department ID.

```
select dept_id, avg(vac_taken)
  from benefits, employee
 where benefits.emp_id = employee.emp_id
    and fiscal_year = 1999
 group by dept_id
 order by dept_id;
```

DEPT_ID	(EXPR)
1100	106.66
1110	160.00
1120	133.33
2200	120.00
2210	115.00
3510	100.00
3520	120.00
3530	120.00
4500	133.33
4600	99.42
5000	84.00
5100	120.00
5200	100.00
6200	86.66

14 rows processed

4. The budget committee needs a list of job titles, names of employees holding those jobs, and current salaries of those employees. They are interested only in jobs offering salaries of more than \$55,000. Order your list by job title and include the job ID.

```
select j.job_id, job_title, emp_lname,
       emp_fname, salary_amount
  from job j, position p, employee e
 where j.job_id = p.job_id
       and p.emp_id = e.emp_id
       and salary_amount > 55000
 order by job_title;
```

JOB_ID	JOB_TITLE	EMP_LNAME	EMP_FNAME	SALARY_AMOUNT
4023	Accountant	Taylor	Cynthia	74776.00
5890	Appraisal Spec	Smith	Michael	66144.00
5890	Appraisal Spec	Brooks	John	68016.00
5110	CUST SER MGR	Bradley	George	56977.80
6011	Manager - Acctng	Bennett	Patricia	94953.52
6004	Manager - HR	Voltmer	Louise	110448.00
6021	Manager - Mktng	Griffin	William	111593.00
4734	Mktng Admin	Robinson	Judith	57824.50
9001	President	Baldwin	James	146432.00
4700	Purch Agnt	Hamel	Marylou	59488.00
4700	Purch Agnt	Johnson	Eleanor	59280.00
5555	Salesperson	Albertini	Joan	76961.00
5555	Salesperson	Parker	Cora	76440.00
5555	Salesperson	Lynn	David	70720.00
4666	Sr Mechanic	Carlson	Thomas	85280.00
8001	Vice President	Mills	Thomas	117832.68

16 rows processed

5. Employee 2004 has just had a review and is due to get a pay increase. The increase is stored as REVIEW_PERCENT in the BENEFITS table. Employee 2004's manager has asked you to show her how much the increase is in dollar amount. To get this information, you need to multiply the current salary by the review percent. Show employee ID, current salary, percent increase, and increase as a dollar amount.

```

select position.emp_id, salary_amount,
       review_percent, (review_percent * salary_amount)
from benefits, position
where "position".emp_id = benefits.emp_id
      and "position".emp_id = 2004
      and fiscal_year = 2000
      and finish_date is null;

```

Notes:

POSITION is the table name **and** an SQL keyword; therefore, when the POSITION table name is used as an identifier, it must be enclosed in double quotation marks.

As an alternative, you can use an alias for the table name. For example:

```

select position.emp_id, salary_amount,
       review_percent, (review_percent * salary_amount)
from benefits b, position p
where p.emp_id = b.emp_id
      and p.emp_id = 2004
      and fiscal_year = 2000
      and finish_date is null;

```

EMP_ID	SALARY_AMOUNT	REVIEW_PERCENT	(EXPR)
2004	59280.00	0.030	1778.40000

1 row processed

B.7 Chapter 7

Exercise 7-1 answer: This is the answer for 7.3.1, “Exercise 7-1” on page 7-5.

Exercises

```
select distinct dept_id
  from employee
 where emp_id in
    (select emp_id
     from benefits
     where (vac_accrued - vac_taken) > 80)
     and fiscal_year = 1999;
```

Exercise 7-2 answer: This is the answer for 7.4.1, “Exercise 7-2” on page 7-7.

```
select emp_id, num_dependents
  from coverage
 where num_dependents >
    (select avg(num_dependents)
     from coverage);
```

Exercise 7-2b answer: This is the answer for 7.4.1.1, “Exercise 7-2B” on page 7-8.

```
select distinct(emp_id), num_dependents
  from coverage
 where num_dependents >
    (select avg(num_dependents)
     from coverage);
```

Exercise 7-3 answer: This is the answer for 7.5.1, “Exercise 7-3” on page 7-11.

```
select distinct dept_id
  from employee
 where exists
    (select *
     from position
     where salary_amount > 50000
     and employee.emp_id = "position".emp_id);
```

Or,

```
select distinct dept_id
  from employee
 where exists
    (select *
     from position p
     where salary_amount > 50000
     and employee.emp_id = p.emp_id);
```

Review answers: These are the answers for 7.7, “Review” on page 7-15.

1. A nested SELECT statement is also known as a **subquery**.
2. A subquery is located in a **WHERE** clause.
3. A subquery must be enclosed in **parentheses**.
4. You use the **EXISTS** predicate to retrieve rows based on the existence of rows in another table.

5. When using an EXISTS predicate, the outer SELECT statement and the subquery are linked by matching **columns** in the WHERE clause in the subquery.
6. You can use an asterisk (*) in the subquery if you are using the **EXISTS** keyword.

Scenario answers: These are the answers for 7.8, “Scenarios” on page 7-16.

1. For tax purposes, the Accounting department needs to keep track of all jobs for which employees earn more than \$65,000. A list of job titles is sufficient. (Use the JOB and POSITION tables.)

```
select job_title
       from job
       where job_id in
             (select job_id from position
              where salary_amount > 65000);
```

```
JOB_TITLE
-----
Accountant
Appraisal Spec
Manager - Acctng
Manager - HR
Manager - Mktng
President
Salesperson
Sr Mechanic
Vice President

9 rows processed
```

2. Upper management is concerned about the equality of salaries within Commonwealth Auto. They need to have a list by name of all jobs for which at least one employee earns less than \$35,000. (Use the JOB and POSITION tables.)

```
select job_title
       from job
       where job_id in
             (select job_id from position
              where salary_amount < 35000);
```

```
JOB_TITLE
-----
Admin Asst
Purch Clerk
Sales Trainee

3 rows processed
```

3. Over the years, lots of department information has been added to the database. The Human Resources department is responsible for this portion of the database and knows that there are some departments still listed for which there are no longer any associated employees. They have asked you for a list showing these departments. Order the list by department ID. (Use the DEPARTMENT and EMPLOYEE tables.)

```
select dept_id
       from department
      where not exists
            (select *
             from employee
             where employee.dept_id = department.dept_id)
      order by dept_id;
```

```
DEPT_ID
-----
    4200
    4900
    6000

3 rows processed
```

B.8 Chapter 8

Exercise 8-1 answer: This is the answer for 8.2.1, “Exercise 8-1” on page 8-4.

```
insert into department
  values (6060, null, 'D09', 'Claims');

select *
  from department
  order by dept_id;
```

Exercise 8-2 answer: This is the answer for 8.2.2, “Exercise 8-2” on page 8-6.

```
insert into department
  values (dept_id, null, 'div_code', 'dept_name');

insert into department
  values (dept_id, null, 'div_code', 'dept_name');

select *
  from department
  order by dept_id;
```

Exercise 8-3 answer: This is the answer for 8.2.3, “Exercise 8-3” on page 8-6.

```
insert into project (proj_id, proj_desc)
  values ('P434', 'Mass Media Campaign Blitz');

select proj_id, proj_desc
  from project
  order by proj_id;
```

Exercise 8-4 answer: This is the answer for 8.3.1, “Exercise 8-4” on page 8-8.

```
select emp_id, vac_accrued
  from benefits
  where fiscal_year = 2000
  order by emp_id;
```

Exercise 8-5 answer: This is the answer for 8.3.2, “Exercise 8-5” on page 8-10.

```
update department
  set dept_name = 'Lost Claims'
  where dept_id = 6060;

select dept_id, dept_name
  from department
  where dept_id = 6060;
```

Exercise 8-6 answer: This is the answer for 8.3.3, “Exercise 8-6” on page 8-10.

```
update employee
  set dept_id = 6200
  where emp_id = 3433;

select emp_id, dept_id
  from employee
  where emp_id = 3433;
```

Exercise 8-7 answer: This is the answer for 8.3.4, “Exercise 8-7” on page 8-11.

```
update employee
  set city = 'Framingham'
  where emp_id in (1034, 3704, 4660);

select emp_id, city
  from employee
  where emp_id in (3433, 8377, 1034);
```

Exercise 8-8 answer: This is the answer for 8.4.1, “Exercise 8-8” on page 8-12.

```
delete from department
  where dept_id = 5050;

select *
  from department
  order by dept_id;
```

Exercise 8-9 answer: This is the answer for 8.4.2, “Exercise 8-9” on page 8-13.

```
delete from department
  where dept_id = 6060;

select *
  from department
  order by dept_id;
```

Review answers: These are the answers for 8.5, “Review” on page 8-14.

1. You use a SELECT statement with INSERT to:
 - a. **Copy specific rows from one table to another**
2. If you don't have a value for every column you are adding to a table, you can:
 - a. **Identify only the columns you are going to insert values into**
 - b. **Use the keyword NULL for the columns where the value is unknown**
3. You can update all rows in a table by:
 - b. **Omitting the WHERE clause**
4. You can update selected rows in a table by:
 - c. **Specifying a search condition in a WHERE clause**
5. You are updating all columns in a table but do not know the specific value to put into one column. You can:
 - c. **Use the keyword NULL for the column where the value is unknown**
6. If you do not have a WHERE clause in a DELETE statement:
 - c. **All the rows are deleted but the table remains**

Appendix C. Table Descriptions

C.1 Table names and descriptions C-3

C.1 Table names and descriptions

ASSIGNMENT

EMP_ID	Unique employee ID
PROJ_ID	ID of project to which consultant is assigned
START_DATE	Date employee was assigned to the project
END_DATE	Date employee completed work on the project

BENEFITS

FISCAL_YEAR	Fiscal year for which this data applies
EMP_ID	Unique employee ID
VAC_ACCRUED	Vacation hours accrued to date
VAC_TAKEN	Vacation hours taken to date
SICK_ACCRUED	Sick days accrued to date
SICK_TAKEN	Sick days taken to date
STOCK_PERCENT	Percentage of earnings allocated to stock purchase
STOCK_AMOUNT	Year-to-date amount deducted for stock purchase
LAST_REVIEW_DATE	Date of last employee review
REVIEW_PERCENT	Percent increase at last review
PROMO_DATE	Date of last promotion
RETIRE_PLAN	Retirement fund identifier: STOCK, BONDS, 401K
RETIRE_PERCENT	Percentage of earnings deducted for retirement
BONUS_AMOUNT	Amount of last bonus
COMP_ACCRUED	Hours of compensation time accrued
COMP_TAKEN	Hours of compensation time taken
EDUC_LEVEL	Level of education: GED, HSDIP, JRCOLL, COLL, MAS, PHD
UNION_ID	Union identification number
UNION_DUES	Amount of dues deducted per pay period

CONSULTANT

CON_ID	Unique consultant ID
CON_FNAME	Consultant's first name
CON_LNAME	Consultant's last name
MANAGER_ID	Employee ID of consultant's manager
DEPT_ID	ID of department to which consultant is assigned
PROJ_ID	ID of project to which consultant is assigned
STREET	Consultant's street address
CITY	Consultant's city
STATE	Consultant's state
ZIP_CODE	Consultant's zip code
PHONE	Consultant's phone
BIRTH_DATE	Birth date
START_DATE	Consultant's date of hire
SS_NUMBER	Social security number
RATE	Hourly rate of pay

COVERAGE

PLAN_CODE	Code of insurance plan providing the coverage
EMP_ID	Unique employee ID
SELECTION_DATE	Date employee selected this insurance plan
TERMINATION_DATE	Date employee terminated this insurance plan; if null, plan is still in force
NUM_DEPENDENTS	Number of dependents covered under this insurance plan

DEPARTMENT

DEPT_ID	Unique department ID
DEPT_HEAD_ID	Employee ID of department head
DIV_CODE	Code of the division to which this department belongs
DEPT_NAME	Department name

DIVISION

DIV_CODE	Unique division ID
DIV_HEAD_ID	Employee ID of division head
DIV_NAME	Division name

EMPLOYEE

EMP_ID	Unique employee ID
MANAGER_ID	Employee ID of employee's manager
EMP_FNAME	Employee's first name
EMP_LNAME	Employee's last name
DEPT_ID	ID of department to which employee is assigned
STREET	Employee's street address
CITY	Employee's city
STATE	Employee's state
ZIP_CODE	Employee's zip code
PHONE	Employee's phone
STATUS	Status of employee: (A) Active; (S) Short-term disability; (L) Long term disability
SS_NUMBER	Social security number
START_DATE	Employee's date of hire
TERMINATION_DATE	Date of termination
BIRTH_DATE	Birth date

EXPERTISE

EMP_ID	Employee ID
SKILL_ID	Skill ID
SKILL_LEVEL	Level of ability in this skill: 01 (low) to 04 (high)
EXP_DATE	Date this level of ability was achieved

INSURANCE_PLAN

PLAN_CODE	Unique plan code for company offering the insurance
COMP_NAME	Name of insurance company
STREET	Street address of insurance company
CITY	City address of insurance company
STATE	State address of insurance company
ZIP_CODE	Zip code of insurance company
PHONE	Telephone number of insurance company
GROUP_NUMBER	Commonwealth's group number for this insurance company
DEDUCT	Dollar amount deductible per year for this insurance plan
MAX_LIFE_BENEFIT	Maximum dollar amount to be paid to insured employee
FAMILY_COST	Amount deducted per paycheck for family coverage
DEP_COST	Additional amount deducted per paycheck per dependent
EFF_DATE	Date this coverage plan becomes effective

JOB

JOB_ID	Unique job ID
JOB_TITLE	Job title
MIN_RATE	Minimum salary/hourly rate for this job
MAX_RATE	Maximum salary/hourly rate for this job
SALARY_IND	Indicator for type of salary: (S) salaried; (H) hourly
NUM_OF_POSITIONS	Total number of positions for this job
NUM_OPEN	Number of positions currently open
EFF_DATE	Date this job became effective
JOB_DESLINE_1	First line of job description
JOB_DESLINE_2	Second line of job description

POSITION

EMP_ID	Employee ID
JOB_ID	Job ID associated with this employee
START_DATE	Date employee began this job
FINISH_DATE	Date employee ended this job (null if current)
HOURLY_RATE	Hourly rate earned while in this job (if hourly position)
SALARY_AMOUNT	Yearly salary earned while in this job (if salaried position)
BONUS_PERCENT	Bonus percent amount for this position (if sales position)
COMM_PERCENT	Commission percent for this position (if sales position)
OVERTIME_RATE	Overtime rate for this position (if hourly position)

PROJECT

PROJ_ID	Unique project ID
PROJ_LEADER_ID	Employee ID of project leader
EST_START_DATE	Estimated date project is to begin
EST_END_DATE	Estimated date project is to end
ACT_START_DATE	Actual date project began
ACT_END_DATE	Actual date project ended
EST_MAN_HOURS	Total number of hours estimated for project
ACT_MAN_HOURS	Actual number of hours required for project
PROJ_DESC	Project description

SKILL

SKILL_ID	Unique skill ID
SKILL_NAME	Skill name
SKILL_DESC	Skill description

Index
