7

SEVEN

Introduction to Structured Query Language (SQL)

In this chapter, you will learn:

- The basic commands and functions of SQL
- How to use SQL for data administration (to create tables, indexes, and views)
- How to use SQL for data manipulation (to add, modify, delete, and retrieve data)
- How to use SQL to query a database for useful information

Preview

In this chapter, you will learn the basics of Structured Query Language (SQL). SQL, pronounced S-Q-L by some and "sequel" by others, is composed of commands that enable users to create database and table structures, perform various types of data manipulation and data administration, and query the database to extract useful information. All relational DBMS software supports SQL, and many software vendors have developed extensions to the basic SQL command set.

Because SQL's vocabulary is simple, the language is relatively easy to learn. Its simplicity is enhanced by the fact that much of its work takes place behind the scenes. For example, a single command creates the complex table structures required to store and manipulate data successfully. Furthermore, SQL is a nonprocedural language; that is, the user specifies what must be done, but not how it is to be done. To issue SQL commands, end users and programmers do not need to know the physical data storage format or the complex activities that take place when a SQL command is executed.

Although quite useful and powerful, SQL is not meant to stand alone in the applications arena. Data entry with SQL is possible but awkward, as are data corrections and additions. SQL itself does not create menus, special report forms, overlays, pop-ups, or any of the other utilities and screen devices that end users usually expect. Instead, those features are available as vendor-supplied enhancements. SQL focuses on data definition (creating tables, indexes, and views) and data manipulation (adding, modifying, deleting, and retrieving data); we will cover these basic functions in this chapter. In spite of its limitations, SQL is a powerful tool for extracting information and managing data.

7.1 INTRODUCTION TO SQL

Ideally, a database language allows you to create database and table structures, to perform basic data management chores (add, delete, and modify), and to perform complex queries designed to transform the raw data into useful information. Moreover, a database language must perform such basic functions with minimal user effort, and its command structure and syntax must be easy to learn. Finally, it must be portable; that is, it must conform to some basic standard so that an individual does not have to relearn the basics when moving from one RDBMS to another. SQL meets those ideal database language requirements well.

SQL functions fit into two broad categories:

- It is a *data definition language (DDL)*: SQL includes commands to create database objects such as tables, indexes, and views, as well as commands to define access rights to those database objects. The data definition commands you will learn in this chapter are listed in Table 7.1.

TABLE 7.1 SQL Data Definition Commands

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows/columns from one or more tables
ALTER TABLE	Modifies a tables definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

- It is a *data manipulation language (DML)*: SQL includes commands to insert, update, delete, and retrieve data within the database tables. The data manipulation commands you will learn in this chapter are listed in Table 7.2.

TABLE 7.2 SQL Data Manipulation Commands

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows

TABLE 7.2 SQL Data Manipulation Commands (continued)

COMMAND OR OPTION	DESCRIPTION
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values
COMPARISON OPERATORS	
=, <, >, <=, >=, <>	Used in conditional expressions
LOGICAL OPERATORS	
AND/OR/NOT	Used in conditional expressions
SPECIAL OPERATORS	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
AGGREGATE FUNCTIONS	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

You will be happy to know that SQL is relatively easy to learn. Its basic command set has a vocabulary of fewer than 100 words. Better yet, SQL is a nonprocedural language: you merely command *what* is to be done; you don't have to worry about *how* it is to be done. The American National Standards Institute (ANSI) prescribes a standard SQL—the current fully approved version is SQL-2003. The ANSI SQL standards are also accepted by the International Organization for Standardization (ISO), a consortium composed of national standards bodies of more than 150 countries. Although adherence to the ANSI/ISO SQL standard is usually required in commercial and government contract database specifications, many RDBMS vendors add their own special enhancements. Consequently, it is seldom possible to move a SQL-based application from one RDBMS to another without making some changes.

However, even though there are several different SQL "dialects," the differences among them are minor. Whether you use Oracle, Microsoft SQL Server, MySQL, IBM's DB2, Microsoft Access, or any other well-established RDBMS, a software manual should be sufficient to get you up to speed if you know the material presented in this chapter.

At the heart of SQL is the query. In Chapter 1, Database Systems, you learned that a query is a spur-of-the-moment question. Actually, in the SQL environment, the word *query* covers both questions and actions. Most SQL queries are used to answer questions such as these: "What products currently held in inventory are priced over $100, and what is the quantity on hand for each of those products?" "How many employees have been hired since January 1, 2008 by each of the company's departments?" However, many SQL queries are used to perform actions such as adding or deleting table rows or changing attribute values within tables. Still other SQL queries create new tables or indexes. In short, for a DBMS, a query is simply a SQL statement that must be executed. But before you can use SQL to query a database, you must define the database environment for SQL with its data definition commands.

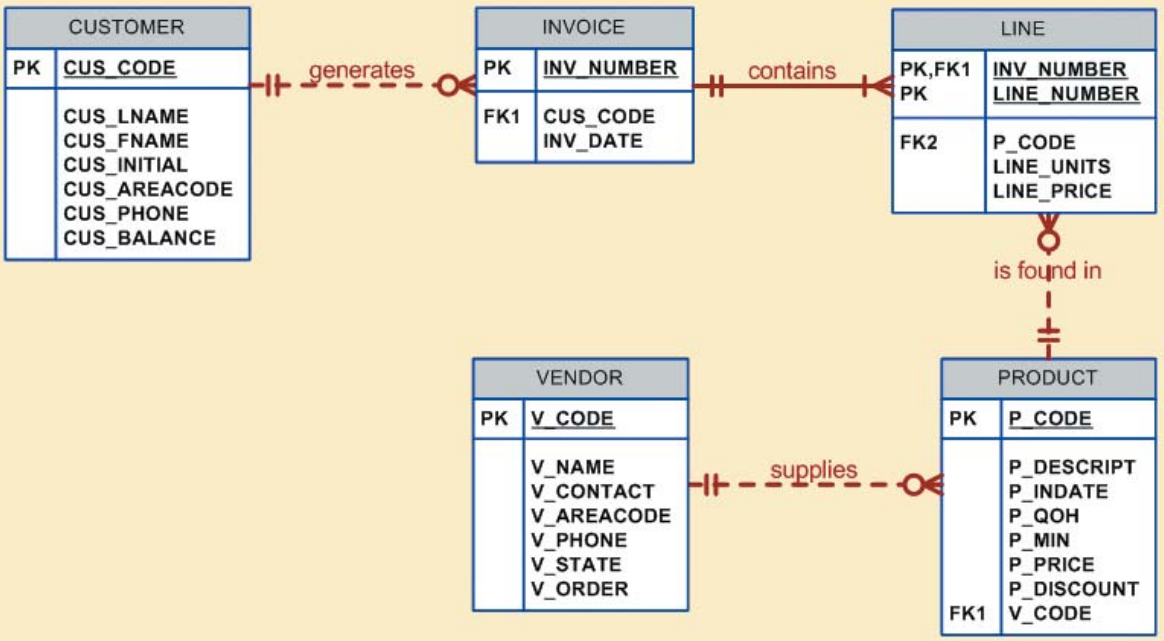
7.2 DATA DEFINITION COMMANDS

Before examining the SQL syntax for creating and defining tables and other elements, let's first examine the simple database model and the database tables that will form the basis for the many SQL examples you'll explore in this chapter.

7.2.1 The Database Model

A simple database composed of the following tables is used to illustrate the SQL commands in this chapter: CUSTOMER, INVOICE, LINE, PRODUCT, and VENDOR. This database model is shown in Figure 7.1.

FIGURE 7.1 The database model



The database model in Figure 7.1 reflects the following business rules:

- A customer may generate many invoices. Each invoice is generated by one customer.
- An invoice contains one or more invoice lines. Each invoice line is associated with one invoice.
- Each invoice line references one product. A product may be found in many invoice lines. (You can sell more than one hammer to more than one customer.)
- A vendor *may* supply many products. Some vendors do not (yet?) supply products. (For example, a vendor list may include *potential* vendors.)
- If a product is vendor-supplied, that product is supplied by only a single vendor.
- Some products are not supplied by a vendor. (For example, some products may be produced in-house or bought on the open market.)

As you can see in Figure 7.1, the database model contains many tables. However, to illustrate the initial set of data definition commands, the focus of attention will be the PRODUCT and VENDOR tables. You will have the opportunity to use the remaining tables later in this chapter and in the problem section.

ONLINE CONTENT

The database model in Figure 7.1 is implemented in the Microsoft Access **Ch07_SaleCo** database located in the Premium Website for this book. (This database contains a few additional tables that are not reflected in Figure 7.1. These tables are used for discussion purposes only.) If you use MS Access, you can use the database supplied online. However, it is strongly suggested that you create your own database structures so you can practice the SQL commands illustrated in this chapter.

SQL script files for creating the tables and loading the data in Oracle and MS SQL Server are also located in the Premium Website. How you connect to your database depends on how the software was installed on your computer. Follow the instructions provided by your instructor or school.

So that you have a point of reference for understanding the effect of the SQL queries, the contents of the PRODUCT and VENDOR tables are listed in Figure 7.2.

FIGURE 7.2 **The VENDOR and PRODUCT tables**

Table name: VENDOR **Database name: Ch07_SaleCo**

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-0092	FL	Y

Table name: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-09	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-09	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-10	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-10	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-09	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-09	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-10	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-10	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-10	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-09	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-10	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-10	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-10	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-10	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-10	18	5	119.95	0.10	25595

Note the following about these tables. (The features correspond to the business rules reflected in the ERD shown in Figure 7.1.)

- The VENDOR table contains vendors who are not referenced in the PRODUCT table. Database designers note that possibility by saying that PRODUCT is *optional* to VENDOR; a vendor may exist without a reference to a product. You examined such optional relationships in detail in Chapter 4, Entity Relationship (ER) Modeling.

- Existing V_CODE values in the PRODUCT table must (and do) have a match in the VENDOR table to ensure referential integrity.
- A few products are supplied factory-direct, a few are made in-house, and a few may have been bought in a warehouse sale. In other words, a product is not necessarily supplied by a vendor. Therefore, VENDOR is optional to PRODUCT.

A few of the conditions just described were made for the sake of illustrating specific SQL features. For example, null V_CODE values were used in the PRODUCT table to illustrate (later) how you can track such nulls using SQL.

7.2.2 Creating the Database

Before you can use a new RDBMS, you must complete two tasks: first, create the database structure, and second, create the tables that will hold the end-user data. To complete the first task, the RDBMS creates the physical files that will hold the database. When you create a new database, the RDBMS automatically creates the data dictionary tables in which to store the metadata and creates a default database administrator. Creating the physical files that will hold the database means interacting with the operating system and the file systems supported by the operating system. Therefore, creating the database structure is the one feature that tends to differ substantially from one RDBMS to another. The good news is that it is relatively easy to create a database structure, regardless of which RDBMS you use.

If you use Microsoft Access, creating the database is simple: start Access, select *File* → *New* → *Blank Database*, specify the folder in which you want to store the database, and then name the database. However, if you work in a database environment typically used by larger organizations, you will probably use an enterprise RDBMS such as Oracle, SQL Server, MySQL, or DB2. Given their security requirements and greater complexity, those database products require a more elaborate database creation process. (See Appendix N, Creating a New Database using Oracle 11g, for an illustration of specific instructions to create a database structure in Oracle.)

You will be relieved to discover that, *with the exception of the database creation process*, most RDBMS vendors use SQL that deviates little from the ANSI standard SQL. For example, most RDBMSs require that each SQL command ends with a semicolon. However, some SQL implementations do not use a semicolon. Important syntax differences among implementations will be highlighted in the Note boxes.

If you are using an enterprise RDBMS, before you can start creating tables you must be authenticated by the RDBMS. **Authentication** is the process through which the DBMS verifies that only registered users may access the database. To be authenticated, you must log on to the RDBMS using a user ID and a password created by the database administrator. In an enterprise RDBMS, every user ID is associated with a database schema.

7.2.3 The Database Schema

In the SQL environment, a **schema** is a group of database objects—such as tables and indexes—that are related to each other. Usually, the schema belongs to a single user or application. A single database can hold multiple schemas belonging to different users or applications. Think of a schema as a logical grouping of database objects, such as tables, indexes, and views. Schemas are useful in that they group tables by owner (or function) and enforce a first level of security by allowing each user to see only the tables that belong to that user.

ANSI SQL standards define a command to create a database schema:

CREATE SCHEMA AUTHORIZATION {creator};

Therefore, if the creator is JONES, use the command:

CREATE SCHEMA AUTHORIZATION JONES;

Most enterprise RDBMSs support that command. However, the command is seldom used directly—that is, from the command line. (When a user is created, the DBMS automatically assigns a schema to that user.) When the DBMS *is* used, the CREATE SCHEMA AUTHORIZATION command must be issued by the user who owns the schema. That is, if you log on as JONES, you can only use CREATE SCHEMA AUTHORIZATION JONES.

For most RDBMSs, the CREATE SCHEMA AUTHORIZATION is optional. That is why this chapter focuses on the ANSI SQL commands required to create and manipulate tables.

7.2.4 Data Types

In the data dictionary in Table 7.3, note particularly the data types selected. Keep in mind that data-type selection is usually dictated by the nature of the data and by the intended use. For example:

- P_PRICE clearly requires some kind of numeric data type; defining it as a character field is not acceptable.
- Just as clearly, a vendor name is an obvious candidate for a character data type. For example, VARCHAR2(35) fits well because vendor names are "variable-length" character strings, and in this case, such strings may be up to 35 characters long.
- At first glance, it might seem logical to select a numeric data type for V_AREACODE because it contains only digits. However, adding and subtracting area codes does not yield meaningful results. Therefore, selecting a character data type is more appropriate. This is true for many common attributes found in business data models. For example, even though zip codes contain all digits, they must be defined as character data because some zip codes begin with the digit zero (0), and a numeric data type would cause the leading zero to be dropped.
- U.S. state abbreviations are always two characters, so CHAR(2) is a logical choice.
- Selecting P_INDATE to be a (Julian) DATE field rather than a character field is desirable because the Julian dates allow you to make simple date comparisons and to perform date arithmetic. For instance, if you have used DATE fields, you can determine how many days there are between them.

If you use DATE fields, you can also determine what the date will be in say, 60 days from a given P_INDATE by using P_INDATE + 60. Or you can use the RDBMS's system date—SYSDATE in Oracle, GETDATE() in MS SQL Server, and Date() in Access—to determine the answer to questions such as, "What will be the date 60 days from today?" For example, you might use SYSDATE + 60 (in Oracle), GETDATE() + 60 (in MS SQL Server), or Date() + 60 (in Access).

Date arithmetic capability is particularly useful in billing. Perhaps you want your system to start charging interest on a customer balance 60 days after the invoice is generated. Such simple date arithmetic would be impossible if you used a character data type.

Data-type selection sometimes requires professional judgment. For example, you must make a decision about the V_CODE's data type as follows:

- If you want the computer to generate new vendor codes by adding 1 to the largest recorded vendor code, you must classify V_CODE as a numeric attribute. (You cannot perform mathematical procedures on character data.) The designation INTEGER will ensure that only the counting numbers (integers) can be used. Most SQL implementations also permit the use of SMALLINT for integer values up to six digits.
- If you do not want to perform mathematical procedures based on V_CODE, you should classify it as a character attribute, even though it is composed entirely of numbers. Character data are "quicker" to process in queries. Therefore, when there is no need to perform mathematical procedures on the attribute, store it as a character attribute.

The first option is used to demonstrate the SQL procedures in this chapter.

TABLE 7.3 Data Dictionary for the CH07_SALECO Database

TABLE NAME	ATTRIBUTE NAME	CONTENTS	TYPE	FORMAT	RANGE*	REQUIRED	PK OR FK	FK REFERENCED TABLE
PRODUCT	P_CODE	Product code	CHAR(10)	XXXXXXXXX	NA	Y	PK	
	P_DESCRIPT	Product description	VARCHAR(35)	Xxxxxxxxxxx	NA	Y		
	P_INDATE	Stocking date	DATE	DD-MON-YYYY	NA	Y		
	P_QOH	Units available	SMALLINT	####	0-9999	Y		
	P_MIN	Minimum units	SMALLINT	####	0-9999	Y		
	P_PRICE	Product price	NUMBER(8,2)	####.##	0.00-9999.00	Y		
	P_DISCOUNT	Discount rate	NUMBER(5,2)	0.##	0.00-0.20	Y		
	V_CODE	Vendor code	INTEGER	###	100-999		FK	VENDOR
VENDOR	V_CODE	Vendor code	INTEGER	#####	1000-9999	Y	PK	
	V_NAME	Vendor name	CHAR(35)	Xxxxxxxxxxxxx	NA	Y		
	V_CONTACT	Contact person	CHAR(25)	Xxxxxxxxxxxxx	NA	Y		
	V_AREACODE	Area code	CHAR(3)	999	NA	Y		
	V_PHONE	Phone number	CHAR(8)	999-9999	NA	Y		
	V_STATE	State	CHAR(2)	XX	NA	Y		
	V_ORDER	Previous order	CHAR(1)	X	Y or N	Y		

FK = Foreign key
PK = Primary key
CHAR = Fixed character length data, 1 to 255 characters
VARCHAR = Variable character length data, 1 to 2,000 characters. VARCHAR is automatically converted to VARCHAR2 in Oracle.
NUMBER = Numeric data. NUMBER(9,2) is used to specify numbers with two decimal places and up to nine digits long, including the decimal places. Some RDBMSs permit the use of a MONEY or a CURRENCY data type.
INT = Integer values only
SMALLINT = Small integer values only
DATE formats vary. Commonly accepted formats are: 'DD-MON-YYYY', 'DD-MON-YY', 'MM/DD/YYYY', and 'MM/DD/YY'
* Not all the ranges shown here will be illustrated in this chapter. However, you can use these constraints to practice writing your own constraints.

When you define the attribute's data type, you must pay close attention to the expected use of the attributes for sorting and data-retrieval purposes. For example, in a real estate application, an attribute that represents the numbers of bathrooms in a home (H_BATH_NUM) could be assigned the CHAR(3) data type because it is highly unlikely the application will do any addition, multiplication, or division with the number of bathrooms. Based on the CHAR(3) data-type definition, valid H_BATH_NUM values would be '2','1','2.5','10'. However, this data-type decision creates potential problems. For example, if an application sorts the homes by number of bathrooms, a query would "see" the value '10' as less than '2', which is clearly incorrect. So you must give some thought to the expected use of the data in order to properly define the attribute data type.

The data dictionary in Table 7.3 contains only a few of the data types supported by SQL. For teaching purposes, the selection of data types is limited to ensure that almost any RDBMS can be used to implement the examples. If your RDBMS is fully compliant with ANSI SQL, it will support many more data types than the ones shown in Table 7.4. And many RDBMSs support data types beyond the ones specified in ANSI SQL.

TABLE 7.4 Some Common SQL Data Types

DATA TYPE	FORMAT	COMMENTS
Numeric	NUMBER(L,D)	The declaration NUMBER(7,2) indicates numbers that will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place. Examples: 12.32, -134.99.
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a *minimum specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.*
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as Smith and Katzenjammer are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. The designation VARCHAR2(25) will let you store characters up to 25 characters long. However, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.
Date	DATE	Stores dates in the Julian date format.

In addition to the data types shown in Table 7.4, SQL supports several other data types, including TIME, TIMESTAMP, REAL, DOUBLE, FLOAT, and intervals such as INTERVAL DAY TO HOUR. Many RDBMSs have also expanded the list to include other types of data, such as LOGICAL, CURRENCY, AutoNumber (Access), and sequence (Oracle). However, because this chapter is designed to introduce the SQL basics, the discussion is limited to the data types summarized in Table 7.4.

7.2.5 Creating Table Structures

Now you are ready to implement the PRODUCT and VENDOR table structures with the help of SQL, using the **CREATE TABLE** syntax shown next.

```
CREATE TABLE tablename (
        column1         data type       [constraint] [,
        column2         data type       [constraint] ] [,
        PRIMARY KEY     (column1        [, column2]) ] [,
        FOREIGN KEY     (column1        [, column2]) REFERENCES tablename] [,
        CONSTRAINT      constraint ] );
```

Online Content

All the SQL commands you will see in this chapter are located in script files in the Premium Website for this book. You can copy and paste the SQL commands into your SQL program. Script files are provided for Oracle and SQL Server users.

To make the SQL code more readable, most SQL programmers use one line per column (attribute) definition. In addition, spaces are used to line up the attribute characteristics and constraints. Finally, both table and attribute names are fully capitalized. Those conventions are used in the following examples that create VENDOR and PRODUCT tables and throughout the book.

Note

SQL SYNTAX

Syntax notation for SQL commands used in this book:

CAPITALS	Required SQL command keywords
italics	An end-user-provided parameter (generally required)
{a | b | ..}	A mandatory parameter; use one option from the list separated by |
[.....]	An optional parameter—anything inside square brackets is optional
Tablename	The name of a table
Column	The name of an attribute in a table
data type	A valid data-type definition
constraint	A valid constraint definition
condition	A valid conditional expression (evaluates to true or false)
columnlist	One or more column names or expressions separated by commas
tablelist	One or more table names separated by commas
conditionlist	One or more conditional expressions separated by logical operators
expression	A simple value (such as 76 or Married) or a formula (such as P_PRICE − 10)

```
CREATE TABLE VENDOR (
V_CODE          INTEGER         NOT NULL   UNIQUE,
V_NAME          VARCHAR(35)     NOT NULL,
V_CONTACT       VARCHAR(15)     NOT NULL,
V_AREACODE      CHAR(3)         NOT NULL,
V_PHONE         CHAR(8)         NOT NULL,
V_STATE         CHAR(2)         NOT NULL,
V_ORDER         CHAR(1)         NOT NULL,
PRIMARY KEY (V_CODE));
```

NOTE

- Because the PRODUCT table contains a foreign key that references the VENDOR table, create the VENDOR table first. (In fact, the M side of a relationship always references the 1 side. Therefore, in a 1:M relationship, you must *always* create the table for the 1 side first.)
- If your RDBMS does not support the VARCHAR2 and FCHAR format, use CHAR.
- Oracle accepts the VARCHAR data type and automatically converts it to VARCHAR2.
- If your RDBMS does not support SINT or SMALLINT, use INTEGER or INT. If INTEGER is not supported, use NUMBER.
- If you use Access, you can use the NUMBER data type, but you cannot use the number delimiters at the SQL level. For example, using NUMBER(8,2) to indicate numbers with up to eight characters and two decimal places is fine in Oracle, but you cannot use it in Access—you must use NUMBER without the delimiters.
- If your RDBMS does not support primary and foreign key designations or the UNIQUE specification, delete them from the SQL code shown here.
- If you use the PRIMARY KEY designation in Oracle, you do not need the NOT NULL and UNIQUE specifications.
- The ON UPDATE CASCADE clause is part of the ANSI standard, but it may not be supported by your RDBMS. In that case, delete the ON UPDATE CASCADE clause.

```
CREATE TABLE PRODUCT (
P_CODE          VARCHAR(10)     NOT NULL   UNIQUE,
P_DESCRIPT      VARCHAR(35)     NOT NULL,
P_INDATE        DATE            NOT NULL,
P_QOH           SMALLINT        NOT NULL,
P_MIN           SMALLINT        NOT NULL,
P_PRICE         NUMBER(8,2)     NOT NULL,
P_DISCOUNT      NUMBER(5,2)     NOT NULL,
V_CODE          INTEGER,
PRIMARY KEY (P_CODE),
FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE);
```

As you examine the preceding SQL table-creating command sequences, note the following features:

- The NOT NULL specifications for the attributes ensure that a data entry will be made. When it is crucial to have the data available, the NOT NULL specification will not allow the end user to leave the attribute empty (with no data entry at all). Because this specification is made at the table level and stored in the data dictionary, application programs can use this information to create the data dictionary validation automatically.
- The UNIQUE specification creates a unique index in the respective attribute. Use it to avoid having duplicated values in a column.

- The primary key attributes contain both a NOT NULL and a UNIQUE specification. Those specifications enforce the entity integrity requirements. If the NOT NULL and UNIQUE specifications are not supported, use PRIMARY KEY without the specifications. (For example, if you designate the PK in MS Access, the NOT NULL and UNIQUE specifications are automatically assumed and are not spelled out.)
- The entire table definition is enclosed in parentheses. A comma is used to separate each table element (attributes, primary key, and foreign key) definition.

NOTE

If you are working with a composite primary key, all of the primary key's attributes are contained within the parentheses and are separated with commas. For example, the LINE table in Figure 7.1 has a primary key that consists of the two attributes INV_NUMBER and LINE_NUMBER. Therefore, you would define the primary key by typing:

```
PRIMARY KEY (INV_NUMBER, LINE_NUMBER),
```

The order of the primary key components is important because the indexing starts with the first-mentioned attribute, then proceeds with the next attribute, and so on. In this example, the line numbers would be ordered within each of the invoice numbers:

INV_NUMBER	LINE_NUMBER
1001	1
1001	2
1002	1
1003	1
1003	2

- The ON UPDATE CASCADE specification ensures that if you make a change in any VENDOR's V_CODE, that change is automatically applied to all foreign key references throughout the system (cascade) to ensure that referential integrity is maintained. (Although the ON UPDATE CASCADE clause is part of the ANSI standard, some RDBMSs, such as Oracle, do not support ON UPDATE CASCADE. If your RDBMS does not support the clause, delete it from the code shown here.)
- An RDBMS will automatically enforce referential integrity for foreign keys. That is, you cannot have an invalid entry in the foreign key column; at the same time, you cannot delete a vendor row as long as a product row references that vendor.
- The command sequence ends with a semicolon. (Remember, your RDBMS may require that you omit the semicolon.)

NOTE

NOTE ABOUT COLUMN NAMES

Do *not* use mathematical symbols such as +, −, and / in your column names; instead, use an underscore to separate words, if necessary. For example, PER-NUM might generate an error message, but PER_NUM is acceptable. Also, do *not* use reserved words. **Reserved words** are words used by SQL to perform specific functions. For example, in some RDBMSs, the column name INITIAL will generate the message invalid column name.

NOTE

NOTE TO ORACLE USERS

When you press the Enter key after typing each line, a line number is automatically generated as long as you do not type a semicolon before pressing the Enter key. For example, Oracle's execution of the CREATE TABLE command will look like this:

```
CREATE TABLE PRODUCT (
  2     P_CODE          VARCHAR2(10)
  3     CONSTRAINT      PRODUCT_P_CODE_PK PRIMARY KEY,
  4     P_DESCRIPT      VARCHAR2(35)      NOT NULL,
  5     P_INDATE        DATE              NOT NULL,
  6     P_QOH           NUMBER            NOT NULL,
  7     P_MIN           NUMBER            NOT NULL,
  8     P_PRICE         NUMBER(8,2)       NOT NULL,
  9     P_DISCOUNT      NUMBER(5,2)       NOT NULL,
 10     V_CODE           NUMBER,
 11     CONSTRAINT      PRODUCT_V_CODE_FK
 12     FOREIGN KEY     V_CODE REFERENCES VENDOR)
 13     ;
```

In the preceding SQL command sequence, note the following:

- The attribute definition for P_CODE starts in line 2 and ends with a comma at the end of line 3.
- The CONSTRAINT clause (line 3) allows you to define and name a constraint in Oracle. You can name the constraint to meet your own naming conventions. In this case, the constraint was named PRODUCT_P_CODE_PK.
- Examples of constraints are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK. For additional details about constraints, see below.
- To define a PRIMARY KEY constraint, you could also use the following syntax: P_CODE VARCHAR2(10) PRIMARY KEY,.
- In this case, Oracle would automatically name the constraint.
- Lines 11 and 12 define a FOREIGN KEY constraint name PRODUCT_V_CODE_FK for the attribute V_CODE. The CONSTRAINT clause is generally used at the end of the CREATE TABLE command sequence.
- *If you do not name the constraints yourself, Oracle will automatically assign a name. Unfortunately, the Oracle-assigned name makes sense only to Oracle, so you will have a difficult time deciphering it later. You should assign a name that makes sense to human beings!*

7.2.6 SQL Constraints

In Chapter 3, The Relational Database Model, you learned that adherence to rules on entity integrity and referential integrity is crucial in a relational database environment. Fortunately, most SQL implementations support both integrity rules. Entity integrity is enforced automatically when the primary key is specified in the CREATE TABLE command sequence. For example, you can create the VENDOR table structure and set the stage for the enforcement of entity integrity rules by using:

```
PRIMARY KEY (V_CODE)
```

In the PRODUCT table's CREATE TABLE sequence, note that referential integrity has been enforced by specifying in the PRODUCT table:

```
FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE
```

That foreign key constraint definition ensures that:

- You cannot delete a vendor from the VENDOR table if at least one product row references that vendor. This is the default behavior for the treatment of foreign keys.
- On the other hand, if a change is made in an existing VENDOR table's V_CODE, that change must be reflected automatically in any PRODUCT table V_CODE reference (ON UPDATE CASCADE). That restriction makes it impossible for a V_CODE value to exist in the PRODUCT table pointing to a nonexistent VENDOR table V_CODE value. In other words, the ON UPDATE CASCADE specification ensures the preservation of referential integrity. (Oracle does not support ON UPDATE CASCADE.)

In general, ANSI SQL permits the use of ON DELETE and ON UPDATE clauses to cover CASCADE, SET NULL, or SET DEFAULT.

ONLINE CONTENT

For a more detailed discussion of the options for the ON DELETE and ON UPDATE clauses, see **Appendix D, Converting an ER Model into a Database Structure,** Section D.2, General Rules Governing Relationships Among Tables. Appendix D is in the Premium Website.

NOTE

NOTE ABOUT REFERENTIAL CONSTRAINT ACTIONS

The support for the referential constraints actions varies from product to product. For example:

- MS Access, SQL Server, and Oracle support ON DELETE CASCADE.
- MS Access and SQL Server support ON UPDATE CASCADE.
- Oracle does not support ON UPDATE CASCADE.
- Oracle supports SET NULL.
- MS Access and SQL Server do not support SET NULL.
- Refer to your product manuals for additional information on referential constraints.

While MS Access does not support ON DELETE CASCADE or ON UPDATE CASCADE at the SQL command-line level, it does support them through the relationship window interface. In fact, whenever you try to establish a relationship between two tables in Access, the relationship window interface will automatically pop up.

Besides the PRIMARY KEY and FOREIGN KEY constraints, the ANSI SQL standard also defines the following constraints:

- The NOT NULL constraint ensures that a column does not accept nulls.
- The UNIQUE constraint ensures that all values in a column are unique.
- The DEFAULT constraint assigns a value to an attribute when a new row is added to a table. The end user may, of course, enter a value other than the default value.
- The CHECK constraint is used to validate data when an attribute value is entered. The CHECK constraint does precisely what its name suggests: it checks to see that a specified condition exists. Examples of such constraints include the following:
  - *The minimum order value must be at least 10.*
  - *The date must be after April 15, 2010.*

If the CHECK constraint is met for the specified attribute (that is, the condition is true), the data are accepted for that attribute. If the condition is found to be false, an error message is generated and the data are not accepted.

Note that the CREATE TABLE command lets you define constraints in two different places:

- When you create the column definition (known as a *column constraint*).
- When you use the CONSTRAINT keyword (known as a *table constraint*).

A column constraint applies to just one column; a table constraint may apply to many columns. Those constraints are supported at varying levels of compliance by enterprise RDBMSs.

In this chapter, Oracle is used to illustrate SQL constraints. For example, note that the following SQL command sequence uses the DEFAULT and CHECK constraints to define the table named CUSTOMER.

```
CREATE TABLE CUSTOMER (
CUS_CODE          NUMBER          PRIMARY KEY,
CUS_LNAME         VARCHAR(15)     NOT NULL,
CUS_FNAME         VARCHAR(15)     NOT NULL,
CUS_INITIAL       CHAR(1),
CUS_AREACODE      CHAR(3)         DEFAULT '615'       NOT NULL
                                  CHECK(CUS_AREACODE IN ('615','713','931')),
CUS_PHONE         CHAR(8)         NOT NULL,
CUS_BALANCE       NUMBER(9,2)     DEFAULT 0.00,
CONSTRAINT CUS_UI1 UNIQUE (CUS_LNAME, CUS_FNAME));
```

In this case, the CUS_AREACODE attribute is assigned a default value of '615'. Therefore, if a new CUSTOMER table row is added and the end user makes no entry for the area code, the '615' value will be recorded. Also note that the CHECK condition restricts the values for the customer's area code to 615, 713, and 931; any other values will be rejected.

It is important to note that the DEFAULT value applies only when new rows are added to a table and then only when no value is entered for the customer's area code. (The default value is not used when the table is modified.) In contrast, the CHECK condition is validated whether a customer row is added *or modified*. However, while the CHECK condition may include any valid expression, it applies only to the attributes in the table being checked. If you want to check for conditions that include attributes in other tables, you must use triggers. (See Chapter 8, Advanced SQL.) Finally, the last line of the CREATE TABLE command sequence creates a unique index constraint (named CUS_UI1) on the customer's last name and first name. The index will prevent the entry of two customers with the same last name and first name. (This index merely illustrates the process. Clearly, it should be possible to have more than one person named John Smith in the CUSTOMER table.)

NOTE

NOTE TO MS ACCESS USERS

MS Access does not accept the DEFAULT or CHECK constraints. However, MS Access will accept the CONSTRAINT CUS_UI1 UNIQUE (CUS_LNAME, CUS_FNAME) line and create the unique index.

In the following SQL command to create the INVOICE table, the DEFAULT constraint assigns a default date to a new invoice, and the CHECK constraint validates that the invoice date is greater than January 1, 2010.

```
CREATE TABLE INVOICE (
INV_NUMBER        NUMBER          PRIMARY KEY,
CUS_CODE          NUMBER          NOT NULL REFERENCES CUSTOMER(CUS_CODE),
INV_DATE          DATE            DEFAULT SYSDATE NOT NULL,
CONSTRAINT INV_CK1 CHECK (INV_DATE > TO_DATE('01-JAN-2010','DD-MON-YYYY')));
```

In this case, notice the following:

- The CUS_CODE attribute definition contains REFERENCES CUSTOMER (CUS_CODE) to indicate that the CUS_CODE is a foreign key. This is another way to define a foreign key.
- The DEFAULT constraint uses the SYSDATE special function. This function always returns today's date.
- The invoice date (INV_DATE) attribute is automatically given today's date (returned by SYSDATE) when a new row is added and no value is given for the attribute.
- A CHECK constraint is used to validate that the invoice date is greater than 'January 1, 2010'. When comparing a date to a manually entered date in a CHECK clause, Oracle requires the use of the TO_DATE function. The TO_DATE function takes two parameters: the literal date and the date format used.

The final SQL command sequence creates the LINE table. The LINE table has a composite primary key (INV_NUMBER, LINE_NUMBER) and uses a UNIQUE constraint in INV_NUMBER and P_CODE to ensure that the same product is not ordered twice in the same invoice.

```
CREATE TABLE LINE (
INV_NUMBER          NUMBER          NOT NULL,
LINE_NUMBER         NUMBER(2,0)     NOT NULL,
P_CODE              VARCHAR(10)     NOT NULL,
LINE_UNITS          NUMBER(9,2)     DEFAULT 0.00        NOT NULL,
LINE_PRICE          NUMBER(9,2)     DEFAULT 0.00        NOT NULL,
PRIMARY KEY (INV_NUMBER, LINE_NUMBER),
FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE ON DELETE CASCADE,
FOREIGN KEY (P_CODE) REFERENCES PRODUCT(P_CODE),
CONSTRAINT LINE_UI1 UNIQUE(INV_NUMBER, P_CODE));
```

In the creation of the LINE table, note that a UNIQUE constraint is added to prevent the duplication of an invoice line. A UNIQUE constraint is enforced through the creation of a unique index. Also note that the ON DELETE CASCADE foreign key action enforces referential integrity. The use of ON DELETE CASCADE is recommended for weak entities to ensure that the deletion of a row in the strong entity automatically triggers the deletion of the corresponding rows in the dependent weak entity. In that case, the deletion of an INVOICE row will automatically delete all of the LINE rows related to the invoice. In the following section, you will learn more about indexes and how to use SQL commands to create them.

7.2.7 SQL Indexes

You learned in Chapter 3 that indexes can be used to improve the efficiency of searches and to avoid duplicate column values. In the previous section, you saw how to declare unique indexes on selected attributes when the table is created. In fact, when you declare a primary key, the DBMS automatically creates a unique index. Even with this feature, you often need additional indexes. The ability to create indexes quickly and efficiently is important. Using the **CREATE INDEX** command, SQL indexes can be created on the basis of any selected attribute. The syntax is:

CREATE [UNIQUE] INDEX *indexname* ON *tablename*(*column1* [, *column2*])

For example, based on the attribute P_INDATE stored in the PRODUCT table, the following command creates an index named P_INDATEX:

```
CREATE INDEX P_INDATEX ON PRODUCT(P_INDATE);
```

SQL does not let you write over an existing index without warning you first, thus preserving the index structure within the data dictionary. Using the UNIQUE index qualifier, you can even create an index that prevents you from using a value that has been used before. Such a feature is especially useful when the index attribute is a candidate key whose values must not be duplicated:

CREATE UNIQUE INDEX P_CODEX ON PRODUCT(P_CODE);

If you now try to enter a duplicate P_CODE value, SQL produces the error message "duplicate value in index." Many RDBMSs, including Access, automatically create a unique index on the PK attribute(s) when you declare the PK.

A common practice is to create an index on any field that is used as a search key, in comparison operations in a conditional expression, or when you want to list rows in a specific order. For example, if you want to create a report of all products by vendor, it would be useful to create an index on the V_CODE attribute in the PRODUCT table. Remember that a vendor can supply many products. Therefore, you should *not* create a UNIQUE index in this case. Better yet, to make the search as efficient as possible, using a composite index is recommended.

Unique composite indexes are often used to prevent data duplication. For example, consider the case illustrated in Table 7.5, in which required employee test scores are stored. (An employee can take a test only once on a given date.) Given the structure of Table 7.5, the PK is EMP_NUM + TEST_NUM. The third test entry for employee 111 meets entity integrity requirements—the combination 111,3 is unique—yet the WEA test entry is clearly duplicated.

TABLE 7.5 A Duplicated Test Record

EMP_NUM	TEST_NUM	TEST_CODE	TEST_DATE	TEST_SCORE
110	1	WEA	15-Jan-2010	93
110	2	WEA	12-Jan-2010	87
111	1	HAZ	14-Dec-2009	91
111	2	WEA	18-Feb-2010	95
111	3	WEA	18-Feb-2010	95
112	1	CHEM	17-Aug-2009	91

Such duplication could have been avoided through the use of a unique composite index, using the attributes EMP_NUM, TEST_CODE, and TEST_DATE:

CREATE UNIQUE INDEX EMP_TESTDEX ON TEST(EMP_NUM, TEST_CODE, TEST_DATE);

By default, all indexes produce results that are listed in ascending order, but you can create an index that yields output in descending order. For example, if you routinely print a report that lists all products ordered by price from highest to lowest, you could create an index named PROD_PRICEX by typing:

CREATE INDEX PROD_PRICEX ON PRODUCT(P_PRICE DESC);

To delete an index, use the **DROP INDEX** command:

DROP INDEX *indexname*

For example, if you want to eliminate the PROD_PRICEX index, type:

DROP INDEX PROD_PRICEX;

After creating the tables and some indexes, you are ready to start entering data. The following sections use two tables (VENDOR and PRODUCT) to demonstrate most of the data manipulation commands.

7.3 DATA MANIPULATION COMMANDS

In this section, you will learn how to use the basic SQL data manipulation commands INSERT, SELECT, COMMIT, UPDATE, ROLLBACK, and DELETE.

7.3.1 Adding Table Rows

SQL requires the use of the **INSERT** command to enter data into a table. The INSERT command's basic syntax looks like this:

INSERT INTO *tablename* VALUES (*value1, value2, ... , valuen*)

Because the PRODUCT table uses its V_CODE to reference the VENDOR table's V_CODE, an integrity violation will occur if those VENDOR table V_CODE values don't yet exist. Therefore, you need to enter the VENDOR rows before the PRODUCT rows. Given the VENDOR table structure defined earlier and the sample VENDOR data shown in Figure 7.2, you would enter the first two data rows as follows:

```
INSERT INTO VENDOR
      VALUES (21225,'Bryson, Inc.','Smithson','615','223-3234','TN','Y');
INSERT INTO VENDOR
      VALUES (21226,'Superloo, Inc.','Flushing','904','215-8995','FL','N');
```

and so on, until all of the VENDOR table records have been entered.

(To see the contents of the VENDOR table, use the SELECT * FROM VENDOR; command.)

The PRODUCT table rows would be entered in the same fashion, using the PRODUCT data shown in Figure 7.2. For example, the first two data rows would be entered as follows, pressing the Enter key at the end of each line:

```
INSERT INTO PRODUCT
      VALUES ('11QER/31','Power painter, 15 psi., 3-nozzle','03-Nov-09',8,5,109.99,0.00,25595);
INSERT INTO PRODUCT
      VALUES ('13-Q2/P2','7.25-in. pwr. saw blade','13-Dec-09',32,15,14.99, 0.05, 21344);
```

(To see the contents of the PRODUCT table, use the SELECT * FROM PRODUCT; command.)

NOTE

Date entry is a function of the date format expected by the DBMS. For example, March 25, 2010 might be shown as 25-Mar-2010 in Access and Oracle, or it might be displayed in other presentation formats in another RDBMS. MS Access requires the use of # delimiters when performing any computations or comparisons based on date attributes, as in P_INDATE >= #25-Mar-10#.

In the preceding data entry lines, observe that:

- The row contents are entered between parentheses. Note that the first character after VALUES is a parenthesis and that the last character in the command sequence is also a parenthesis.
- Character (string) and date values must be entered between apostrophes (').
- Numerical entries are *not* enclosed in apostrophes.
- Attribute entries are separated by commas.
- A value is required for each column in the table.

This version of the INSERT commands adds one table row at a time.

Inserting Rows with Null Attributes

Thus far, you have entered rows in which all of the attribute values are specified. But what do you do if a product does not have a vendor or if you don't yet know the vendor code? In those cases, you would want to leave the vendor code null. To enter a null, use the following syntax:

```
INSERT INTO PRODUCT
      VALUES ('BRT-345','Titanium drill bit','18-Oct-09', 75, 10, 4.50, 0.06, NULL);
```

Incidentally, note that the NULL entry is accepted only because the V_CODE attribute is optional—the NOT NULL declaration was not used in the CREATE TABLE statement for this attribute.

Inserting Rows with Optional Attributes

There might be occasions when more than one attribute is optional. Rather than declaring each attribute as NULL in the INSERT command, you can indicate just the attributes that have required values. You do that by listing the attribute names inside parentheses after the table name. For the purpose of this example, assume that the only required attributes for the PRODUCT table are P_CODE and P_DESCRIPT:

```
INSERT INTO PRODUCT(P_CODE, P_DESCRIPT) VALUES ('BRT-345','Titanium drill bit');
```

7.3.2 Saving Table Changes

Any changes made to the table contents are not saved on disk until you close the database, close the program you are using, or use the **COMMIT** command. If the database is open and a power outage or some other interruption occurs before you issue the COMMIT command, your changes will be lost and only the original table contents will be retained. The syntax for the COMMIT command is:

```
COMMIT [WORK]
```

The COMMIT command permanently saves *all* changes—such as rows added, attributes modified, and rows deleted—made to any table in the database. Therefore, if you intend to make your changes to the PRODUCT table permanent, it is a good idea to save those changes by using:

```
COMMIT;
```

NOTE

NOTE TO MS ACCESS USERS
MS Access doesn't support the COMMIT command because it automatically saves changes after the execution of each SQL command.

However, the COMMIT command's purpose is not just to save changes. In fact, the ultimate purpose of the COMMIT and ROLLBACK commands (see Section 7.3.5) is to ensure database update integrity in transaction management. (You will see how such issues are addressed in Chapter 10, Transaction Management and Concurrency Control.)

7.3.3 Listing Table Rows

The **SELECT** command is used to list the contents of a table. The syntax of the SELECT command is as follows:

SELECT *columnlist* FROM *tablename*

The *columnlist* represents one or more attributes, separated by commas. You could use the * (asterisk) as a wildcard character to list all attributes. A **wildcard character** is a symbol that can be used as a general substitute for other characters or commands. For example, to list all attributes and all rows of the PRODUCT table, use:

```
SELECT * FROM PRODUCT;
```

Figure 7.3 shows the output generated by that command. (Figure 7.3 shows all of the rows in the PRODUCT table that serve as the basis for subsequent discussions. If you entered only the PRODUCT table's first two records, as shown in the preceding section, the output of the preceding SELECT command would show only the rows you entered. Don't worry about the difference between your SELECT output and the output shown in Figure 7.3. When you complete the work in this section, you will have created and populated your VENDOR and PRODUCT tables with the correct rows for use in future sections.)

FIGURE 7.3 **The contents of the PRODUCT table**

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-09	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-09	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-10	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-10	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-09	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-09	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-10	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-10	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-10	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-09	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-10	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-10	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-10	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-10	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-10	18	5	119.95	0.10	25595

NOTE

Your listing may not be in the order shown in Figure 7.3. The listings shown in the figure are the result of system-controlled primary-key-based index operations. You will learn later how to control the output so that it conforms to the order you have specified.

NOTE

NOTE TO ORACLE USERS

Some SQL implementations (such as Oracle's) cut the attribute labels to fit the width of the column. However, Oracle lets you set the width of the display column to show the complete attribute name. You can also change the display format, regardless of how the data are stored in the table. For example, if you want to display dollar symbols and commas in the P_PRICE output, you can declare:

```
COLUMN P_PRICE FORMAT $99,999.99
```

to change the output 12347.67 to $12,347.67.

In the same manner, to display only the first 12 characters of the P_DESCRIPT attribute, use:

```
COLUMN P_DESCRIPT FORMAT A12 TRUNCATE
```

Although SQL commands can be grouped together on a single line, complex command sequences are best shown on separate lines, with space between the SQL command and the command's components. Using that formatting convention makes it much easier to see the components of the SQL statements, making it easy to trace the SQL logic, and if necessary, to make corrections. The number of spaces used in the indention is up to you. For example, note the following format for a more complex statement:

```
SELECT    P_CODE, P_DESCRIPT, P_INDATE, P_QOH, P_MIN, P_PRICE, P_DISCOUNT, V_CODE
FROM      PRODUCT;
```

When you run a SELECT command on a table, the RDBMS returns a set of one or more rows that have the same characteristics as a relational table. In addition, the SELECT command lists all rows from the table you specified in the FROM clause. This is a very important characteristic of SQL commands. By default, most SQL data manipulation commands operate over an entire table (or relation). That is why SQL commands are said to be *set-oriented* commands. A SQL set-oriented command works over a set of rows. The set may include one or more columns and zero or more rows from one or more tables.

7.3.4 Updating Table Rows

Use the **UPDATE** command to modify data in a table. The syntax for this command is:

```
UPDATE    tablename
SET       columnname = expression [, columnname = expression]
[WHERE    conditionlist ];
```

For example, if you want to change P_INDATE from December 13, 2009, to January 18, 2010, in the second row of the PRODUCT table (see Figure 7.3), use the primary key (13-Q2/P2) to locate the correct (second) row. Therefore, type:

```
UPDATE    PRODUCT
SET       P_INDATE = '18-JAN-2010'
WHERE     P_CODE = '13-Q2/P2';
```

If more than one attribute is to be updated in the row, separate the corrections with commas:

```
UPDATE    PRODUCT
SET       P_INDATE = '18-JAN-2010', P_PRICE = 17.99, P_MIN = 10
WHERE     P_CODE = '13-Q2/P2';
```

What would have happened if the previous UPDATE command had not included the WHERE condition? The P_INDATE, P_PRICE, and P_MIN values would have been changed in *all* rows of the PRODUCT table. Remember, the UPDATE command is a set-oriented operator. Therefore, if you don't specify a WHERE condition, the UPDATE command will apply the changes to *all* rows in the specified table.

Confirm the correction(s) by using this SELECT command to check the PRODUCT table's listing:

```
SELECT * FROM PRODUCT;
```

7.3.5 Restoring Table Contents

If you have not yet used the COMMIT command to store the changes permanently in the database, you can restore the database to its previous condition with the **ROLLBACK** command. ROLLBACK undoes any changes since the last COMMIT command and brings the data back to the values that existed before the changes were made. To restore the data to their "prechange" condition, type:

```
ROLLBACK;
```

and then press the Enter key. Use the SELECT statement again to see that the ROLLBACK did, in fact, restore the data to their original values.

COMMIT and ROLLBACK work only with data manipulation commands that are used to add, modify, or delete table rows. For example, assume that you perform these actions:

1. CREATE a table called SALES.
2. INSERT 10 rows in the SALES table.
3. UPDATE two rows in the SALES table.
4. Execute the ROLLBACK command.

Will the SALES table be removed by the ROLLBACK command? No, the ROLLBACK command will undo *only* the results of the INSERT and UPDATE commands. All data definition commands (CREATE TABLE) are automatically committed to the data dictionary and cannot be rolled back. The COMMIT and ROLLBACK commands are examined in greater detail in Chapter 10.

NOTE

NOTE TO MS ACCESS USERS
MS Access does not support the ROLLBACK command.

Some RDBMSs, such as Oracle, automatically COMMIT data changes when issuing data definition commands. For example, if you had used the CREATE INDEX command after updating the two rows in the previous example, all previous changes would have been committed automatically; doing a ROLLBACK afterward wouldn't have undone anything. *Check your RDBMS manual to understand these subtle differences.*

7.3.6 DELETING TABLE ROWS

It is easy to delete a table row using the **DELETE** statement; the syntax is:

```
DELETE FROM     tablename
[WHERE          conditionlist ];
```

For example, if you want to delete from the PRODUCT table the product that you added earlier whose code (P_CODE) is 'BRT-345', use:

```
DELETE FROM     PRODUCT
WHERE           P_CODE = 'BRT-345';
```

In that example, the primary key value lets SQL find the exact record to be deleted. However, deletions are not limited to a primary key match; any attribute may be used. For example, in your PRODUCT table, you will see that there are several products for which the P_MIN attribute is equal to 5. Use the following command to delete all rows from the PRODUCT table for which the P_MIN is equal to 5:

```
DELETE FROM     PRODUCT
WHERE           P_MIN = 5;
```

Check the PRODUCT table's contents again to verify that all products with P_MIN equal to 5 have been deleted.

Finally, remember that DELETE is a set-oriented command. And keep in mind that the WHERE condition is optional. Therefore, if you do not specify a WHERE condition, *all* rows from the specified table will be deleted!

7.3.7 Inserting Table Rows with a Select Subquery

You learned in Section 7.3.1 how to use the INSERT statement to add rows to a table. In that section, you added rows one at a time. In this section, you will learn how to add multiple rows to a table, using another table as the source of the data. The syntax for the INSERT statement is:

INSERT INTO *tablename* SELECT *columnlist* FROM *tablename*;

In that case, the INSERT statement uses a SELECT subquery. A **subquery**, also known as a **nested query** or an **inner query**, is a query that is embedded (or nested) inside another query. The inner query is always executed first by the RDBMS. Given the previous SQL statement, the INSERT portion represents the outer query, and the SELECT portion represents the subquery. You can nest queries (place queries inside queries) many levels deep; in every case, the output of the inner query is used as the input for the outer (higher-level) query. In Chapter 8 you will learn more about the various types of subqueries.

The values returned by the SELECT subquery should match the attributes and data types of the table in the INSERT statement. If the table into which you are inserting rows has one date attribute, one number attribute, and one character attribute, the SELECT subquery should return one or more rows in which the first column has date values, the second column has number values, and the third column has character values.

Online Content

Before you execute the commands in the following sections, you **MUST** do the following:

- If you are using Oracle, run the **sqlintrodbinit.sql** script file in the Premium Website to create all tables and load the data in the database.
- If you are using Access, copy the original **Ch07_SaleCo.mbd** file from the Premium Website.

7.4 SELECT QUERIES

In this section, you will learn how to fine-tune the SELECT command by adding restrictions to the search criteria. SELECT, coupled with appropriate search conditions, is an incredibly powerful tool that enables you to transform data into information. For example, in the following sections, you will learn how to create queries that can be used to answer questions such as these: "What products were supplied by a particular vendor?" "Which products are priced below $10?" "How many products supplied by a given vendor were sold between January 5, 2010 and March 20, 2010?"

7.4.1 Selecting Rows with Conditional Restrictions

You can select partial table contents by placing restrictions on the rows to be included in the output. This is done by using the WHERE clause to add conditional restrictions to the SELECT statement. The following syntax enables you to specify which rows to select:

SELECT *columnlist*
FROM *tablelist*
[WHERE *conditionlist*];

The SELECT statement retrieves all rows that match the specified condition(s)—also known as the *conditional criteria*—you specified in the WHERE clause. The *conditionlist* in the WHERE clause of the SELECT statement is represented by one or more conditional expressions, separated by logical operators. The WHERE clause is optional.

If no rows match the specified criteria in the WHERE clause, you see a blank screen or a message that tells you that no rows were retrieved. For example, the query:

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM      PRODUCT
WHERE     V_CODE = 21344;
```

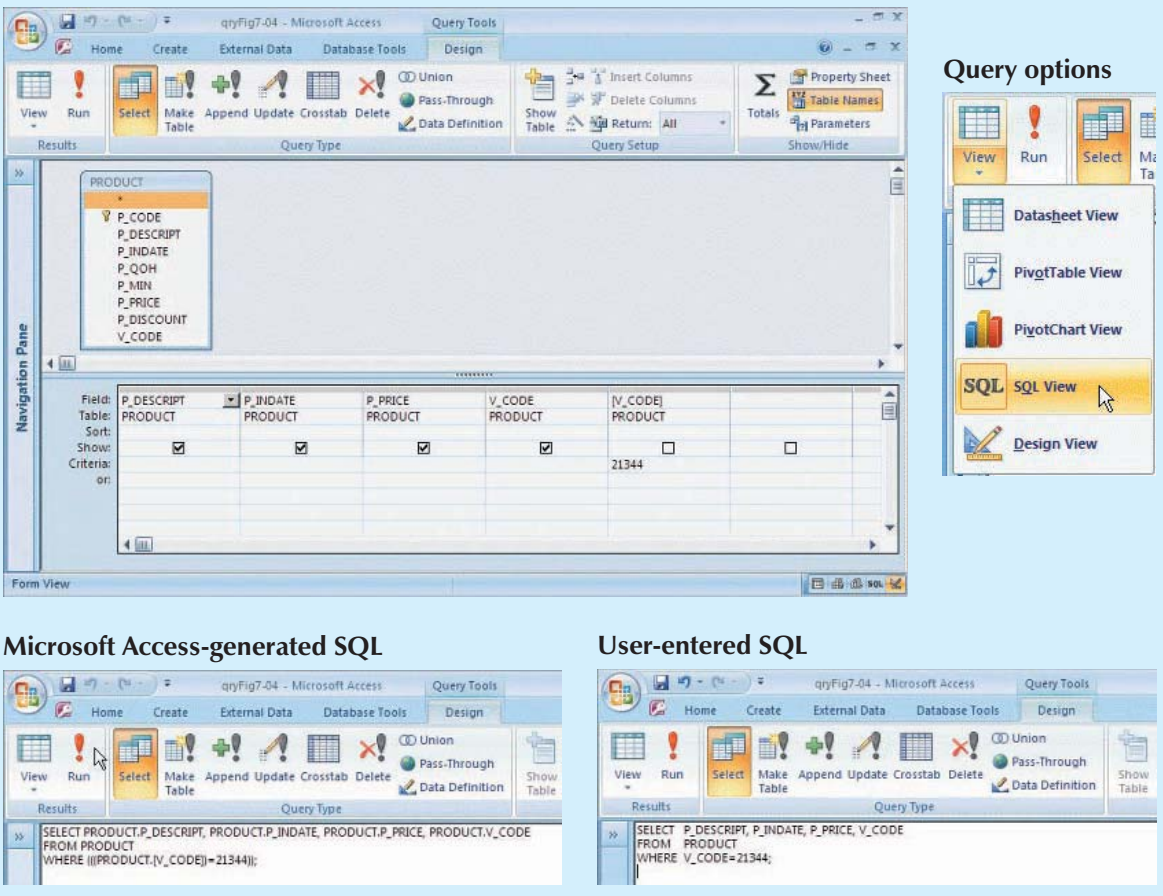
returns the description, date, and price of products with a vendor code of 21344, as shown in Figure 7.4.

FIGURE 7.4 Selected PRODUCT table attributes for VENDOR code 21344

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
7.25-in. pwr. saw blade	13-Dec-09	14.99	21344
9.00-in. pwr. saw blade	13-Nov-09	17.49	21344
Rat-tail file, 1/8-in. fine	15-Dec-09	4.99	21344

MS Access users can use the Access QBE (query by example) query generator. Although the Access QBE generates its own "native" version of SQL, you can also elect to type standard SQL in the Access SQL window, as shown at the bottom of Figure 7.5. Figure 7.5 shows the Access QBE screen, the SQL window's QBE-generated SQL, and the listing of the modified SQL.

FIGURE 7.5 The Microsoft Access QBE and its SQL



Numerous conditional restrictions can be placed on the selected table contents. For example, the comparison operators shown in Table 7.6 can be used to restrict output.

NOTE

NOTE TO MS ACCESS USERS

The MS Access QBE interface automatically designates the data source by using the table name as a prefix. You will discover later that the table name prefix is used to avoid ambiguity when the same column name appears in multiple tables. For example, both the VENDOR and the PRODUCT tables contain the V_CODE attribute. Therefore, if both tables are used (as they would be in a join), the source of the V_CODE attribute must be specified.

TABLE 7.6 Comparison Operators

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

The following example uses the "not equal to" operator:

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM      PRODUCT
WHERE     V_CODE <> 21344;
```

The output, shown in Figure 7.6, lists all of the rows for which the vendor code is *not* 21344.

Note that, in Figure 7.6, rows with nulls in the V_CODE column (see Figure 7.3) are not included in the SELECT command's output.

FIGURE 7.6 Selected PRODUCT table attributes for VENDOR codes other than 21344

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
Power painter, 15 psi., 3-nozzle	03-Nov-09	109.99	25595
Hrd. cloth, 1/4-in., 2x50	15-Jan-10	39.95	23119
Hrd. cloth, 1/2-in., 3x50	15-Jan-10	43.99	23119
B&D jigsaw, 12-in. blade	30-Dec-09	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-09	99.87	24288
B&D cordless drill, 1/2-in.	20-Jan-10	38.95	25595
Claw hammer	20-Jan-10	9.95	21225
Hicut chain saw, 16 in.	07-Feb-10	256.99	24288
1.25-in. metal screw, 25	01-Mar-10	6.99	21225
2.5-in. wd. screw, 50	24-Feb-10	8.45	21231
Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-10	119.95	25595

FIGURE 7.7 Selected PRODUCT table attributes with a P_PRICE restriction

P_DESCRIPT	P_QOH	P_MIN	P_PRICE
Claw hammer	23	10	9.95
Rat-tail file, 1/8-in. fine	43	20	4.99
PVC pipe, 3.5-in., 8-ft	188	75	5.87
1.25-in. metal screw, 25	172	75	6.99
2.5-in. wd. screw, 50	237	100	8.45

The command sequence:

```
SELECT    P_DESCRIPT, P_QOH, P_MIN, P_PRICE
FROM      PRODUCT
WHERE     P_PRICE <= 10;
```

yields the output shown in Figure 7.7.

Using Comparison Operators on Character Attributes

Because computers identify all characters by their (numeric) American Standard Code for Information Interchange (ASCII) codes, comparison operators may even be used to place restrictions on character-based attributes. Therefore, the command:

```
SELECT    P_CODE, P_DESCRIPT, P_QOH, P_MIN,
          P_PRICE
FROM      PRODUCT
WHERE     P_CODE < '1558-QW1';
```

would be correct and would yield a list of all rows in which the P_CODE is alphabetically less than 1558-QW1. (Because the

ASCII code value for the letter *B* is greater than the value of the letter *A*, it follows that *A* is less than *B*.) Therefore, the output will be generated as shown in Figure 7.8.

FIGURE 7.8 Selected PRODUCT table attributes: the ASCII code effect

P_CODE	P_DESCRIPT	P_QOH	P_MIN	P_PRICE
11QER/31	Power painter, 15 psi., 3-nozzle	8	5	109.99
13-Q2/P2	7.25-in. pwr. saw blade	32	15	14.99
14-Q1/L3	9.00-in. pwr. saw blade	18	12	17.49
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15	8	39.95

String (character) comparisons are made from left to right. This left-to-right comparison is especially useful when attributes such as names are to be compared. For example, the string "Ardmore" would be judged *greater than* the string "Aarenson" but *less than* the string "Brown"; such results may be used to generate alphabetical listings like those found in a phone directory. If the characters 0–9 are stored as strings, the same left-to-right string comparisons can lead to apparent anomalies. For example, the ASCII code for the character "5" is, as expected, *greater than* the ASCII code for the character "4." Yet the same "5" will also be judged *greater than* the string "44" because the *first* character in the string "44" is less than the string "5." For that reason, you may get some unexpected results from comparisons when dates or other numbers are stored in character format. This also applies to date comparisons. For example, the left-to-right ASCII character comparison would force the conclusion that the date "01/01/2010" occurred *before* "12/31/2009." Because the leftmost character "0" in "01/01/2010" is *less than* the leftmost character "1" in "12/31/2009," "01/01/2010" is *less than* "12/31/2009." Naturally, if date strings are stored in a yyyy/mm/dd format, the comparisons will yield appropriate results, but this is a nonstandard date presentation. That's why all current RDBMSs support "date" data types; you should use them. In addition, using "date" data types gives you the benefit of date arithmetic.

Using Comparison Operators on Dates

Date procedures are often more software-specific than other SQL procedures. For example, the query to list all of the rows in which the inventory stock dates occur on or after January 20, 2010 will look like this:

```
SELECT    P_DESCRIPT, P_QOH, P_MIN, P_PRICE, P_INDATE
FROM      PRODUCT
WHERE     P_INDATE >= '20-Jan-2010';
```

(Remember that MS Access users must use the # delimiters for dates. For example, you would use #20-Jan-10# in the above WHERE clause.) The date-restricted output is shown in Figure 7.9.

FIGURE 7.9 Selected PRODUCT table attributes: date restriction

P_DESCRIPT	P_QOH	P_MIN	P_PRICE	P_INDATE
B&D cordless drill, 1/2-in.	12	5	38.95	20-Jan-10
Claw hammer	23	10	9.95	20-Jan-10
Hicut chain saw, 16 in.	11	5	256.99	07-Feb-10
PVC pipe, 3.5-in., 8-ft	188	75	5.87	20-Feb-10
1.25-in. metal screw, 25	172	75	6.99	01-Mar-10
2.5-in. wd. screw, 50	237	100	8.45	24-Feb-10

Using Computed Columns and Column Aliases

Suppose that you want to determine the total value of each of the products currently held in inventory. Logically, that determination requires the multiplication of each product's quantity on hand by its current price. You can accomplish this task with the following command:

```
SELECT    P_DESCRIPT, P_QOH, P_PRICE, P_QOH *
          P_PRICE
FROM      PRODUCT;
```

FIGURE 7.10 SELECT statement with a computed column

P_DESCRIPT	P_QOH	P_PRICE	Expr1
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

Entering that SQL command in Access generates the output shown in Figure 7.10.

SQL accepts any valid expressions (or formulas) in the computed columns. Such formulas can contain any valid mathematical operators and functions that are applied to attributes in any of the tables specified in the FROM clause of the SELECT statement. Note also that Access automatically adds an Expr label to all computed columns. (The first computed column would be labeled Expr1; the second, Expr2; and so on.) Oracle uses the actual formula text as the label for the computed column.

To make the output more readable, the SQL standard permits the use of aliases for any column in a SELECT statement. An **alias** is an alternative name given to a column or table in any SQL statement.

For example, you can rewrite the previous SQL statement as:

```
SELECT    P_DESCRIPT, P_QOH, P_PRICE, P_QOH * P_PRICE AS TOTVALUE
FROM      PRODUCT;
```

The output of that command is shown in Figure 7.11.

FIGURE 7.11 SELECT statement with a computed column and an alias

P_DESCRIPT	P_QOH	P_PRICE	TOTVALUE
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

You could also use a computed column, an alias, and date arithmetic in a single query. For example, assume that you want to get a list of out-of-warranty products that have been stored more than 90 days. In that case, the P_INDATE is at least 90 days less than the current (system) date. The MS Access version of this query is:

```
SELECT    P_CODE, P_INDATE, DATE() - 90 AS
          CUTDATE
FROM      PRODUCT
WHERE     P_INDATE <= DATE() - 90;
```

The Oracle version of the same query is shown here:

```
SELECT    P_CODE, P_INDATE, SYSDATE - 90 AS
          CUTDATE
FROM      PRODUCT
WHERE     P_INDATE <= SYSDATE - 90;
```

Note that DATE() and SYSDATE are special functions that return the current date in MS Access and Oracle, respectively. You can use the DATE() and SYSDATE functions anywhere a date literal is expected, such as in the value list of an INSERT statement, in an UPDATE statement when changing the value of a date attribute, or in a SELECT statement as shown here. Of course, the previous query output would change based on the current date.

Suppose that a manager wants a list of all products, the dates they were received, and the warranty expiration date (90 days from when the product was received). To generate that list, type:

```
SELECT    P_CODE, P_INDATE, P_INDATE + 90 AS EXPDATE
FROM      PRODUCT;
```

Note that you can use all arithmetic operators with date attributes as well as with numeric attributes.

7.4.2 Arithmetic Operators: The Rule of Precedence

As you saw in the previous example, you can use arithmetic operators with table attributes in a column list or in a conditional expression. In fact, SQL commands are often used in conjunction with the arithmetic operators shown in Table 7.7.

TABLE 7.7 The Arithmetic Operators

ARITHMETIC OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power of (some applications use ** instead of ^)

Do not confuse the multiplication symbol (*) with the wildcard symbol used by some SQL implementations, such as MS Access; the latter is used only in string comparisons, while the former is used in conjunction with mathematical procedures.

As you perform mathematical operations on attributes, remember the rules of precedence. As the name suggests, the **rules of precedence** are the rules that establish the order in which computations are completed. For example, note the order of the following computational sequence:

1. Perform operations within parentheses.
2. Perform power operations.
3. Perform multiplications and divisions.
4. Perform additions and subtractions.

The application of the rules of precedence will tell you that 8 + 2 * 5 = 8 + 10 = 18, but (8 + 2) * 5 = 10 * 5 = 50. Similarly, 4 + 5^2 * 3 = 4 + 25 * 3 = 79, but (4 + 5)^2 * 3 = 81 * 3 = 243, while the operation expressed by (4 + 5^2) * 3 yields the answer (4 + 25) * 3 = 29 * 3 = 87.

7.4.3 Logical Operators: AND, OR, and NOT

In the real world, a search of data normally involves multiple conditions. For example, when you are buying a new house, you look for a certain area, a certain number of bedrooms, bathrooms, stories, and so on. In the same way, SQL allows you to include multiple conditions in a query through the use of logical operators. The logical operators are AND, OR, and NOT. For example, if you want a list of the table contents for either the V_CODE = 21344 **or** the V_CODE = 24288, you can use the **OR** operator, as in the following command sequence:

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM      PRODUCT
WHERE     V_CODE = 21344 OR V_CODE = 24288;
```

That command generates the six rows shown in Figure 7.12 that match the logical restriction.

FIGURE 7.12 Selected PRODUCT table attributes: the logical OR

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
7.25-in. pwr. saw blade	13-Dec-09	14.99	21344
9.00-in. pwr. saw blade	13-Nov-09	17.49	21344
B&D jigsaw, 12-in. blade	30-Dec-09	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-09	99.87	24288
Rat-tail file, 1/8-in. fine	15-Dec-09	4.99	21344
Hicut chain saw, 16 in.	07-Feb-10	256.99	24288

The logical **AND** has the same SQL syntax requirement. The following command generates a list of all rows for which P_PRICE is less than $50 and for which P_INDATE is a date occurring after January 15, 2010:

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE,
          V_CODE
FROM      PRODUCT
WHERE     P_PRICE < 50
AND       P_INDATE > '15-Jan-2010';
```

This command will produce the output shown in Figure 7.13.

FIGURE 7.13 Selected PRODUCT table attributes: the logical AND

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D cordless drill, 1/2-in.	20-Jan-10	38.95	25595
Claw hammer	20-Jan-10	9.95	21225
PVC pipe, 3.5-in., 8-ft	20-Feb-10	5.87	
1.25-in. metal screw, 25	01-Mar-10	6.99	21225
2.5-in. wd. screw, 50	24-Feb-10	8.45	21231

You can combine the logical OR with the logical AND to place further restrictions on the output. For example, suppose that you want a table listing for the following conditions:

- The P_INDATE is after January 15, 2010, and the P_PRICE is less than $50.
- Or the V_CODE is 24288.

The required listing can be produced by using:

```
SELECT    P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM      PRODUCT
WHERE     (P_PRICE < 50 AND P_INDATE > '15-Jan-2010')
OR        V_CODE = 24288;
```

Note the use of parentheses to combine logical restrictions. Where you place the parentheses depends on how you want the logical restrictions to be executed. Conditions listed within parentheses are always executed first. The preceding query yields the output shown in Figure 7.14.

FIGURE 7.14 Selected PRODUCT table attributes: the logical AND and OR

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D jigsaw, 12-in. blade	30-Dec-09	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-09	99.87	24288
B&D cordless drill, 1/2-in.	20-Jan-10	38.95	25595
Claw hammer	20-Jan-10	9.95	21225
Hicut chain saw, 16 in.	07-Feb-10	256.99	24288
PVC pipe, 3.5-in., 8-ft	20-Feb-10	5.87	
1.25-in. metal screw, 25	01-Mar-10	6.99	21225
2.5-in. wd. screw, 50	24-Feb-10	8.45	21231

Note that the three rows with the V_CODE = 24288 are included regardless of the P_INDATE and P_PRICE entries for those rows.

The use of the logical operators OR and AND can become quite complex when numerous restrictions are placed on the query. In fact, a specialty field in mathematics known as **Boolean algebra** is dedicated to the use of logical operators.

The logical operator **NOT** is used to negate the result of a conditional expression. That is, in SQL, all conditional expressions evaluate to true or false. If an expression is true,

the row is selected; if an expression is false, the row is not selected. The NOT logical operator is typically used to find the rows that *do not* match a certain condition. For example, if you want to see a listing of all rows for which the vendor code is not 21344, use the command sequence:

```
SELECT    *
FROM      PRODUCT
WHERE     NOT (V_CODE = 21344);
```

Note that the condition is enclosed in parentheses; that practice is optional, but it is highly recommended for clarity. The logical NOT can be combined with AND and OR.

NOTE

If your SQL version does not support the logical NOT, you can generate the required output by using the condition:

```
WHERE V_CODE <> 21344
```

If your version of SQL does not support <>, use:

```
WHERE V_CODE != 21344
```

7.4.4 SPECIAL OPERATORS

ANSI-standard SQL allows the use of special operators in conjunction with the WHERE clause. These special operators include:

BETWEEN: Used to check whether an attribute value is within a range

IS NULL: Used to check whether an attribute value is null

LIKE: Used to check whether an attribute value matches a given string pattern

IN: Used to check whether an attribute value matches any value within a value list

EXISTS: Used to check whether a subquery returns any rows

The BETWEEN Special Operator

If you use software that implements a standard SQL, the operator BETWEEN may be used to check whether an attribute value is within a range of values. For example, if you want to see a listing for all products whose prices are between $50 and $100, use the following command sequence:

```
SELECT    *
FROM      PRODUCT
WHERE     P_PRICE BETWEEN 50.00 AND 100.00;
```

NOTE

NOTE TO ORACLE USERS
When using the BETWEEN special operator, always specify the lower range value first. If you list the higher range value first, Oracle will return an empty result set.

If your DBMS does not support BETWEEN, you can use:

```
SELECT    *
FROM      PRODUCT
WHERE     P_PRICE > 50.00 AND P_PRICE < 100.00;
```

The IS NULL Special Operator

Standard SQL allows the use of IS NULL to check for a null attribute value. For example, suppose that you want to list all products that do not have a vendor assigned (V_CODE is null). Such a null entry could be found by using the command sequence:

```
SELECT    P_CODE, P_DESCRIPT, V_CODE
FROM      PRODUCT
WHERE     V_CODE IS NULL;
```

Similarly, if you want to check a null date entry, the command sequence is:

```
SELECT    P_CODE, P_DESCRIPT, P_INDATE
FROM      PRODUCT
WHERE     P_INDATE IS NULL;
```

Note that SQL uses a special operator to test for nulls. Why? Couldn't you just enter a condition such as "V_CODE = NULL"? No. Technically, NULL is not a "value" the way the number 0 (zero) or the blank space is, but instead a NULL is a special property of an attribute that represents precisely the absence of any value.

The LIKE Special Operator

The LIKE special operator is used in conjunction with wildcards to find patterns within string attributes. Standard SQL allows you to use the percent sign (%) and underscore (_) wildcard characters to make matches when the entire string is not known:

- % means any and all *following* or preceding characters are eligible. For example,
  'J%' includes Johnson, Jones, Jernigan, July, and J-231Q.
  'Jo%' includes Johnson and Jones.
  '%n' includes Johnson and Jernigan.
- _ means any *one* character may be substituted for the underscore. For example,
  '_23-456-6789' includes 123-456-6789, 223-456-6789, and 323-456-6789.
  '_23-_56-678_' includes 123-156-6781, 123-256-6782, and 823-956-6788.
  '_o_es' includes Jones, Cones, Cokes, totes, and roles.

NOTE

Some RDBMSs, such as Microsoft Access, use the wildcard characters * and ? instead of % and _.

For example, the following query would find all VENDOR rows for contacts whose last names begin with *Smith*.

```
SELECT    V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      VENDOR
WHERE     V_CONTACT LIKE 'Smith%';
```

If you check the original VENDOR data in Figure 7.2 again, you'll see that this SQL query yields three records: two Smiths and one Smithson.

Keep in mind that most SQL implementations yield case-sensitive searches. For example, Oracle will not yield a result that includes *Jones* if you use the wildcard search delimiter 'jo%' in a search for last names. The reason is that *Jones* begins with a capital *J*, and your wildcard search starts with a lowercase *j*. On the other hand, MS Access searches are not case sensitive.

For example, suppose that you typed the following query in Oracle:

```
SELECT    V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      VENDOR
WHERE     V_CONTACT LIKE 'SMITH%';
```

No rows will be returned because character-based queries may be case sensitive. That is, an uppercase character has a different ASCII code than a lowercase character, causing *SMITH, Smith,* and *smith* to be evaluated as different (unequal) entries. Because the table contains no vendor whose last name begins with (uppercase) *SMITH,* the (uppercase) 'SMITH%' used in the query cannot be matched. Matches can be made only when the query entry is written exactly like the table entry.

Some RDBMSs, such as Microsoft Access, automatically make the necessary conversions to eliminate case sensitivity. Others, such as Oracle, provide a special UPPER function to convert both table and query character entries to uppercase. (The conversion is done in the computer's memory only; the conversion has no effect on how the value is actually stored in the table.) So if you want to avoid a no-match result based on case sensitivity, and if your RDBMS allows the use of the UPPER function, you can generate the same results by using the query:

```
SELECT    V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      VENDOR
WHERE     UPPER(V_CONTACT) LIKE 'SMITH%';
```

The preceding query produces a list that includes all rows containing a last name that begins with *Smith*, regardless of uppercase or lowercase letter combinations such as *Smith, smith,* and *SMITH*.

The logical operators may be used in conjunction with the special operators. For instance, the query:

```
SELECT    V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      VENDOR
WHERE     V_CONTACT NOT LIKE 'Smith%';
```

will yield an output of all vendors whose names do not start with *Smith.*

Suppose that you do not know whether a person's name is spelled Johnson or Johnsen. The wildcard character _ lets you find a match for either spelling. The proper search would be instituted by the query:

```
SELECT    *
FROM      VENDOR
WHERE     V_CONTACT LIKE 'Johns_n';
```

Thus, the wildcards allow you to make matches when only approximate spellings are known. Wildcard characters may be used in combinations. For example, the wildcard search based on the string '_l%' can yield the strings Al, Alton, Elgin, Blakeston, blank, bloated, and eligible.

The IN Special Operator

Many queries that would require the use of the logical OR can be more easily handled with the help of the special operator IN. For example, the query:

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE = 21344
OR          V_CODE = 24288;
```

can be handled more efficiently with:

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN (21344, 24288);
```

Note that the IN operator uses a value list. All of the values in the list must be of the same data type. Each of the values in the value list is compared to the attribute—in this case, V_CODE. If the V_CODE value matches any of the values in the list, the row is selected. In this example, the rows selected will be only those in which the V_CODE is either 21344 or 24288.

If the attribute used is of a character data type, the list values must be enclosed in single quotation marks. For instance, if the V_CODE had been defined as CHAR(5) when the table was created, the preceding query would have read:

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN ('21344', '24288');
```

The IN operator is especially valuable when it is used in conjunction with subqueries. For example, suppose that you want to list the V_CODE and V_NAME of only those vendors who provide products. In that case, you could use a subquery within the IN operator to automatically generate the value list. The query would be:

```
SELECT      V_CODE, V_NAME
FROM        VENDOR
WHERE       V_CODE IN (SELECT V_CODE FROM PRODUCT);
```

The preceding query will be executed in two steps:

1. The inner query or subquery will generate a list of V_CODE values from the PRODUCT tables. Those V_CODE values represent the vendors who supply products.
2. The IN operator will compare the values generated by the subquery to the V_CODE values in the VENDOR table and will select only the rows with matching values—that is, the vendors who provide products.

The IN special operator will receive additional attention in Chapter 8, where you will learn more about subqueries.

The EXISTS Special Operator

The EXISTS special operator can be used whenever there is a requirement to execute a command based on the result of another query. That is, if a subquery returns any rows, run the main query; otherwise, don't. For example, the following query will list all vendors, but only if there are products to order:

```
SELECT      *
FROM        VENDOR
WHERE       EXISTS (SELECT * FROM PRODUCT WHERE P_QOH <= P_MIN);
```

The EXISTS special operator is used in the following example to list all vendors, but only if there are products with the quantity on hand, less than double the minimum quantity:

```
SELECT      *
FROM        VENDOR
WHERE       EXISTS (SELECT * FROM PRODUCT WHERE P_QOH < P_MIN * 2);
```

The EXISTS special operator will receive additional attention in Chapter 8, where you will learn more about subqueries.

7.5 ADDITIONAL DATA DEFINITION COMMANDS

In this section, you will learn how to change (alter) table structures by changing attribute characteristics and by adding columns. Then you will learn how to do advanced data updates to the new columns. Finally, you will learn how to copy tables or parts of tables and how to delete tables.

All changes in the table structure are made by using the **ALTER TABLE** command, followed by a keyword that produces the specific change you want to make. Three options are available: ADD, MODIFY, and DROP. You use ADD to add a column, MODIFY to change column characteristics, and DROP to delete a column from a table. Most RDBMSs do not allow you to delete a column (unless the column does not contain any values) because such an action might delete crucial data that are used by other tables. The basic syntax to add or modify columns is:

ALTER TABLE *tablename*
    {ADD | MODIFY} (*columnname datatype* [{ADD | MODIFY} *columnname datatype*]) ;

The ALTER TABLE command can also be used to add table constraints. In those cases, the syntax would be:

ALTER TABLE *tablename*
    ADD *constraint* [ADD *constraint*] ;

where *constraint* refers to a constraint definition similar to those you learned in Section 7.2.6.

You could also use the ALTER TABLE command to remove a column or table constraint. The syntax would be as follows:

ALTER TABLE *tablename*
    DROP{PRIMARY KEY | COLUMN *columnname* | CONSTRAINT *constraintname* };

Notice that when removing a constraint, you need to specify the name given to the constraint. That is one reason why you should always name your constraints in your CREATE TABLE or ALTER TABLE statement.

7.5.1 Changing a Column's Data Type

Using the ALTER syntax, the (integer) V_CODE in the PRODUCT table can be changed to a character V_CODE by using:

```
ALTER TABLE PRODUCT
    MODIFY (V_CODE CHAR(5));
```

Some RDBMSs, such as Oracle, do not let you change data types unless the column to be changed is empty. For example, if you want to change the V_CODE field from the current number definition to a character definition, the above command will yield an error message, because the V_CODE column already contains data. The error message is easily explained. Remember that the V_CODE in PRODUCT references the V_CODE in VENDOR. If you change the V_CODE data type, the data types don't match, and there is a referential integrity violation, which triggers the error message. If the V_CODE column does not contain data, the preceding command sequence will produce the

expected table structure alteration (if the foreign key reference was not specified during the creation of the PRODUCT table).

7.5.2 Changing a Column's Data Characteristics

If the column to be changed already contains data, you can make changes in the column's characteristics if those changes do not alter the data *type*. For example, if you want to increase the width of the P_PRICE column to nine digits, use the command:

```
ALTER TABLE PRODUCT
      MODIFY (P_PRICE DECIMAL(9,2));
```

If you now list the table contents, you can see that the column width of P_PRICE has increased by one digit.

NOTE

Some DBMSs impose limitations on when it's possible to change attribute characteristics. For example, Oracle lets you increase (but not decrease) the size of a column. The reason for this restriction is that an attribute modification will affect the integrity of the data in the database. In fact, some attribute changes can be done only when there are no data in any rows for the affected attribute.

7.5.3 Adding a Column

You can alter an existing table by adding one or more columns. In the following example, you add the column named P_SALECODE to the PRODUCT table. (This column will be used later to determine whether goods that have been in inventory for a certain length of time should be placed on special sale.)

Suppose that you expect the P_SALECODE entries to be 1, 2, or 3. Because there will be no arithmetic performed with the P_SALECODE, the P_SALECODE will be classified as a single-character attribute. Note the inclusion of all required information in the following ALTER command:

```
ALTER TABLE PRODUCT
      ADD (P_SALECODE CHAR(1));
```

ONLINE CONTENT

If you are using the MS Access databases provided in the Premium Website, you can track each of the updates in the following sections. For example, look at the copies of the PRODUCT table in the **Ch07_SaleCo** database, one named Product_2 and one named PRODUCT_3. Each of the two copies includes the new P_SALECODE column. If you want to see the *cumulative* effect of all UPDATE commands, you can continue using the PRODUCT table with the P_SALECODE modification and all of the changes you will make in the following sections. (You might even want to use both options, first to examine the individual effects of the update queries and then to examine the cumulative effects.)

When adding a column, be careful not to include the NOT NULL clause for the new column. Doing so will cause an error message; if you add a new column to a table that already has rows, the existing rows will default to a value of null for the new column. Therefore, it is not possible to add the NOT NULL clause for this new column. (You can, of course, add the NOT NULL clause to the table structure after all of the data for the new column have been entered and the column no longer contains nulls.)

7.5.4 Dropping a Column

Occasionally, you might want to modify a table by deleting a column. Suppose that you want to delete the V_ORDER attribute from the VENDOR table. To accomplish that, you would use the following command:

```
ALTER TABLE VENDOR
      DROP COLUMN V_ORDER;
```

Again, some RDBMSs impose restrictions on attribute deletion. For example, you may not drop attributes that are involved in foreign key relationships, nor may you delete an attribute of a table that contains only that one attribute.

7.5.5 Advanced Data Updates

To make data entries in an existing row's columns, SQL allows the UPDATE command. The UPDATE command updates only data in existing rows. For example, to enter the P_SALECODE value '2' in the fourth row, use the UPDATE command together with the primary key P_CODE '1546-QQ2'. Enter the value by using the command sequence:

```
UPDATE      PRODUCT
SET         P_SALECODE = '2'
WHERE       P_CODE = '1546-QQ2';
```

Subsequent data can be entered the same way, defining each entry location by its primary key (P_CODE) and its column location (P_SALECODE). For example, if you want to enter the P_SALECODE value '1' for the P_CODE values '2232/QWE' and '2232/QTY', you use:

```
UPDATE      PRODUCT
SET         P_SALECODE = '1'
WHERE       P_CODE IN ('2232/QWE', '2232/QTY');
```

If your RDBMS does not support IN, use the following command:

```
UPDATE      PRODUCT
SET         P_SALECODE = '1'
WHERE       P_CODE = '2232/QWE' OR P_CODE = '2232/QTY';
```

The results of your efforts can be checked by using:

```
SELECT      P_CODE, P_DESCRIPT, P_INDATE, P_PRICE, P_SALECODE
FROM        PRODUCT;
```

Although the UPDATE sequences just shown allow you to enter values into specified table cells, the process is very cumbersome. Fortunately, if a relationship can be established between the entries and the existing columns, the relationship can be used to assign values to their appropriate slots. For example, suppose that you want to place sales codes based on the P_INDATE into the table, using the following schedule:

P_INDATE	P_SALECODE
before December 25, 2009	2
between January 16, 2010, and February 10, 2010	1

Using the PRODUCT table, the following two command sequences make the appropriate assignments:

```
UPDATE      PRODUCT
SET         P_SALECODE = '2'
WHERE       P_INDATE < '25-Dec-2009';
```

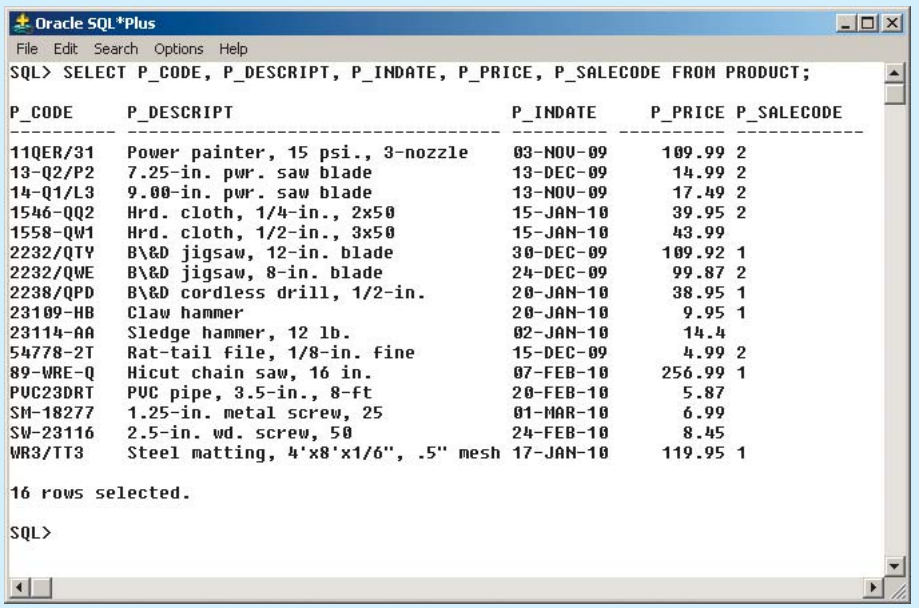
```
UPDATE      PRODUCT
SET         P_SALECODE = '1'
WHERE       P_INDATE >= '16-Jan-2010' AND P_INDATE <='10-Feb-2010';
```

To check the results of those two command sequences, use:

```
SELECT      P_CODE, P_DESCRIPT, P_INDATE, P_PRICE, P_SALECODE
FROM        PRODUCT;
```

If you have made *all* of the updates shown in this section using Oracle, your PRODUCT table should look like Figure 7.15. *Make sure that you issue a COMMIT statement to save these changes.*

FIGURE 7.15 **The cumulative effect of the multiple updates in the PRODUCT table (Oracle)**



The arithmetic operators are particularly useful in data updates. For example, if the quantity on hand in your PRODUCT table has dropped below the minimum desirable value, you'll order more of the product. Suppose, for example, you have ordered 20 units of product 2232/QWE. When the 20 units arrive, you'll want to add them to inventory, using:

```
UPDATE      PRODUCT
SET         P_QOH = P_QOH + 20
WHERE       P_CODE = '2232/QWE';
```

If you want to add 10 percent to the price for all products that have current prices below $50, you can use:

```
UPDATE      PRODUCT
SET         P_PRICE = P_PRICE * 1.10
WHERE       P_PRICE < 50.00;
```

If you are using Oracle, issue a ROLLBACK command to undo the changes made by the last two UPDATE statements.

NOTE

If you fail to roll back the changes of the preceding UPDATE queries, the output of the subsequent queries will not match the results shown in the figures. Therefore:

- If you are using Oracle, use the ROLLBACK command to restore the database to its previous state.
- If you are using Access, copy the original **Ch07_SaleCo.mdb** file from the Premium Website for this book.

7.5.6 Copying Parts of Tables

As you will discover in later chapters on database design, sometimes it is necessary to break up a table structure into several component parts (or smaller tables). Fortunately, SQL allows you to copy the contents of selected table columns so that the data need not be reentered manually into the newly created table(s). For example, if you want to copy P_CODE, P_DESCRIPT, P_PRICE, and V_CODE from the PRODUCT table to a new table named PART, you create the PART table structure first, as follows:

```
CREATE TABLE PART(
PART_CODE          CHAR(8)          NOT NULL          UNIQUE,
PART_DESCRIPT      CHAR(35),
PART_PRICE         DECIMAL(8,2),
V_CODE             INTEGER,
PRIMARY KEY (PART_CODE));
```

Note that the PART column names need not be identical to those of the original table and that the new table need not have the same number of columns as the original table. In this case, the first column in the PART table is PART_CODE, rather than the original P_CODE found in the PRODUCT table. And the PART table contains only four columns rather than the seven columns found in the PRODUCT table. However, column characteristics must match; you cannot copy a character-based attribute into a numeric structure and vice versa.

Next, you need to add the rows to the new PART table, using the PRODUCT table rows. To do that, you use the INSERT command you learned in Section 7.3.7. The syntax is:

```
INSERT INTO    target_tablename[(target_columnlist)]
SELECT         source_columnlist
FROM           source_tablename;
```

Note that the target column list is required if the source column list doesn't match all of the attribute names and characteristics of the target table (including the order of the columns). Otherwise, you do not need to specify the target column list. In this example, you must specify the target column list in the INSERT command below because the column names of the target table are different:

```
INSERT INTO PART    (PART_CODE, PART_DESCRIPT, PART_PRICE, V_CODE)
SELECT              P_CODE, P_DESCRIPT, P_PRICE, V_CODE FROM PRODUCT;
```

The contents of the PART table can now be examined by using the query:

```
SELECT      * FROM PART;
```

to generate the new PART table's contents, shown in Figure 7.16.

FIGURE 7.16 PART table attributes copied from the PRODUCT table

PART_CODE	PART_DESCRIPT	PART_PRICE	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	109.99	25595
13-Q2/P2	7.25-in. pwr. saw blade	14.99	21344
14-Q1/L3	9.00-in. pwr. saw blade	17.49	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	39.95	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	43.99	23119
2232/QTY	B&D jigsaw, 12-in. blade	109.92	24288
2232/QWE	B&D jigsaw, 8-in. blade	99.87	24288
2238/QPD	B&D cordless drill, 1/2-in.	38.95	25595
23109-HB	Claw hammer	9.95	21225
23114-AA	Sledge hammer, 12 lb.	14.4	
54778-2T	Rat-tail file, 1/8-in. fine	4.99	21344
89-WRE-Q	Hicut chain saw, 16 in.	256.99	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	5.87	
SM-18277	1.25-in. metal screw, 25	6.99	21225
SW-23116	2.5-in. wd. screw, 50	8.45	21231
WR3/TT3	Steel matting, 4'x8'x1/8", .5" mesh	119.95	25595

SQL also provides another way to rapidly create a new table based on selected columns and rows of an existing table. In this case, the new table will copy the attribute names, data characteristics, and rows of the original table. The Oracle version of the command is:

```
CREATE TABLE PART AS
SELECT      P_CODE AS PART_CODE, P_DESCRIPT
            AS PART_DESCRIPT,
            P_PRICE AS PART_PRICE, V_CODE
FROM        PRODUCT;
```

If the PART table already exists, Oracle will not let you overwrite the existing table. To run this command, you must first delete the existing PART table. (See Section 7.5.8.)

The MS Access version of this command is:

```
SELECT      P_CODE AS PART_CODE, P_DESCRIPT AS PART_DESCRIPT,
            P_PRICE AS PART_PRICE, V_CODE INTO PART
FROM        PRODUCT;
```

If the PART table already exists, MS Access will ask if you want to delete the existing table and continue with the creation of the new PART table.

The SQL command just shown creates a new PART table with PART_CODE, PART_DESCRIPT, PART_PRICE, and V_CODE columns. In addition, all of the data rows (for the selected columns) will be copied automatically. *However, note that no entity integrity (primary key) or referential integrity (foreign key) rules are automatically applied to the new table.* In the next section, you will learn how to define the PK to enforce entity integrity and the FK to enforce referential integrity.

7.5.7 Adding Primary and Foreign Key Designations

When you create a new table based on another table, the new table does not include integrity rules from the old table. In particular, there is no primary key. To define the primary key for the new PART table, use the following command:

```
ALTER TABLE   PART
      ADD     PRIMARY KEY (PART_CODE);
```

Aside from the fact that the integrity rules are not automatically transferred to a new table that derives its data from one or more other tables, several other scenarios could leave you without entity and referential integrity. For example, you might have forgotten to define the primary and foreign keys when you created the original tables. Or if you imported tables from a different database, you might have discovered that the importing procedure did not transfer the integrity rules. In any case, you can reestablish the integrity rules by using the ALTER command. For example, if the PART table's foreign key has not yet been designated, it can be designated by:

```
ALTER TABLE   PART
      ADD     FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

Alternatively, if neither the PART table's primary key nor its foreign key has been designated, you can incorporate both changes at once, using:

```
ALTER TABLE   PART
      ADD     PRIMARY KEY (PART_CODE)
      ADD     FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

Even composite primary keys and multiple foreign keys can be designated in a single SQL command. For example, if you want to enforce the integrity rules for the LINE table shown in Figure 7.1, you can use:

```
ALTER TABLE   LINE
      ADD     PRIMARY KEY (INV_NUMBER, LINE_NUMBER)
      ADD     FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE
      ADD     FOREIGN KEY (PROD_CODE) REFERENCES PRODUCT;
```

7.5.8 Deleting a Table from the Database

A table can be deleted from the database using the **DROP TABLE** command. For example, you can delete the PART table you just created with:

```
DROP TABLE PART;
```

You can drop a table only if that table is not the "one" side of any relationship. If you try to drop a table otherwise, the RDBMS will generate an error message indicating that a foreign key integrity violation has occurred.

7.6 ADDITIONAL SELECT QUERY KEYWORDS

One of the most important advantages of SQL is its ability to produce complex free-form queries. The logical operators that were introduced earlier to update table contents work just as well in the query environment. In addition, SQL provides useful functions that count, find minimum and maximum values, calculate averages, and so on. Better yet, SQL allows the user to limit queries to only those entries that have no duplicates or entries whose duplicates can be grouped.

7.6.1 Ordering a Listing

The **ORDER BY** clause is especially useful when the listing order is important to you. The syntax is:

```
SELECT      columnlist
FROM        tablelist
[WHERE      conditionlist ]
[ORDER BY   columnlist [ASC | DESC] ] ;
```

Although you have the option of declaring the order type—ascending or descending—the default order is ascending. For example, if you want the contents of the PRODUCT table listed by P_PRICE in ascending order, use:

```
SELECT      P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM        PRODUCT
ORDER BY    P_PRICE;
```

The output is shown in Figure 7.17. Note that ORDER BY yields an ascending price listing.

Comparing the listing in Figure 7.17 to the actual table contents shown earlier in Figure 7.2, you will see that in Figure 7.17, the lowest-priced product is listed first, followed by the next lowest-priced product, and so on. However, although ORDER BY produces a sorted output, the actual table contents are unaffected by the ORDER BY command.

To produce the list in descending order, you would enter:

```
SELECT      P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM        PRODUCT
ORDER BY    P_PRICE DESC;
```

FIGURE 7.17 **Selected PRODUCT table attributes: ordered by (ascending) P_PRICE**

P_CODE	P_DESCRIPT	P_INDATE	P_PRICE
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-09	4.99
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-10	5.87
SM-18277	1.25-in. metal screw, 25	01-Mar-10	6.99
SW-23116	2.5-in. wd. screw, 50	24-Feb-10	8.45
23109-HB	Claw hammer	20-Jan-10	9.95
23114-AA	Sledge hammer, 12 lb	02-Jan-10	14.40
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	14.99
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-09	17.49
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-10	38.95
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-10	39.95
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-10	43.99
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-09	99.87
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-09	109.92
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-09	109.99
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-10	119.95
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-10	256.99

Ordered listings are used frequently. For example, suppose that you want to create a phone directory. It would be helpful if you could produce an ordered sequence (last name, first name, initial) in three stages:

1. ORDER BY last name.
2. Within the last names, ORDER BY first name.
3. Within the first and last names, ORDER BY middle initial.

Such a multilevel ordered sequence is known as a **cascading order sequence**, and it can be created easily by listing several attributes, separated by commas, after the ORDER BY clause.

The cascading order sequence is the basis for any telephone directory. To illustrate a cascading order sequence, use the following SQL command on the EMPLOYEE table:

```
SELECT    EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_AREACODE, EMP_PHONE
FROM      EMPLOYEE
ORDER BY  EMP_LNAME, EMP_FNAME, EMP_INITIAL;
```

That command yields the results shown in Figure 7.18.

FIGURE 7.18 **Telephone list query results**

EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_AREACODE	EMP_PHONE
Brandon	Marie	G	901	882-0845
Diante	Jorge	D	615	890-4567
Genkazi	Leighla	W	901	569-0093
Johnson	Edward	E	615	898-4387
Jones	Anne	M	615	898-3456
Kolmycz	George	D	615	324-5456
Lange	John	P	901	504-4430
Lewis	Rhonda	G	615	324-4472
Saranda	Hermine	R	615	324-5505
Smith	George	A	615	890-2984
Smith	George	K	901	504-3339
Smith	Jeanine	K	615	324-7883
Smythe	Melanie	P	615	324-9006
Vandam	Rhett		901	675-8993
Washington	Rupert	E	615	890-4925
Wiesenbach	Paul	R	615	897-4358
Williams	Robert	D	615	890-3220

The ORDER BY clause is useful in many applications, especially because the DESC qualifier can be invoked. For example, listing the most recent items first is a standard procedure. Typically, invoice due dates are listed in descending order. Or if you want to examine budgets, it's probably useful to list the largest budget line items first.

You can use the ORDER BY clause in conjunction with other SQL commands, too. For example, note the use of restrictions on date and price in the following command sequence:

```
SELECT      P_DESCRIPT, V_CODE, P_INDATE, P_PRICE
FROM        PRODUCT
WHERE       P_INDATE < '21-Jan-2010'
AND         P_PRICE <= 50.00
ORDER BY    V_CODE, P_PRICE DESC;
```

The output is shown in Figure 7.19. Note that within each V_CODE, the P_PRICE values are in descending order.

FIGURE 7.19 A query based on multiple restrictions

P_DESCRIPT	V_CODE	P_INDATE	P_PRICE
B&D cordless drill, 1/2-in.	25595	20-Jan-10	38.95
Hrd. cloth, 1/2-in., 3x50	23119	15-Jan-10	43.99
Hrd. cloth, 1/4-in., 2x50	23119	15-Jan-10	39.95
9.00-in. pwr. saw blade	21344	13-Nov-09	17.49
7.25-in. pwr. saw blade	21344	13-Dec-09	14.99
Rat-tail file, 1/8-in. fine	21344	15-Dec-09	4.99
Claw hammer	21225	20-Jan-10	9.95
Sledge hammer, 12 lb.		02-Jan-10	14.40

7.6.2 Listing Unique Values

How many *different* vendors are currently represented in the PRODUCT table? A simple listing (SELECT) is not very useful if the table contains several thousand rows and you have to sift through the vendor codes manually. Fortunately, SQL's **DISTINCT** clause produces a list of only those values that are different from one another. For example, the command:

```
SELECT      DISTINCT V_CODE
FROM        PRODUCT;
```

yields only the different (distinct) vendor codes (V_CODE) that are encountered in the PRODUCT table, as shown in Figure 7.20. Note that the first output row shows the null. (By default, Access places the null V_CODE at the top of the list, while Oracle places it at the bottom. The placement of nulls does not affect the list contents. In Oracle, you could use ORDER BY V_CODE NULLS FIRST to place nulls at the top of the list.)

NOTE

If the ordering column has nulls, they are listed either first or last, depending on the RDBMS.

The ORDER BY clause must always be listed last in the SELECT command sequence.

FIGURE 7.20 A listing of distinct (different) V_CODE values in the PRODUCT table

V_CODE
21225
21231
21344
23119
24288
25595

7.6.3 Aggregate Functions

SQL can perform various mathematical summaries for you, such as counting the number of rows that contain a specified condition, finding the minimum or maximum values for some specified attribute, summing the values in a specified column, and averaging the values in a specified column. Those aggregate functions are shown in Table 7.8.

TABLE 7.8 Some Basic SQL Aggregate Functions

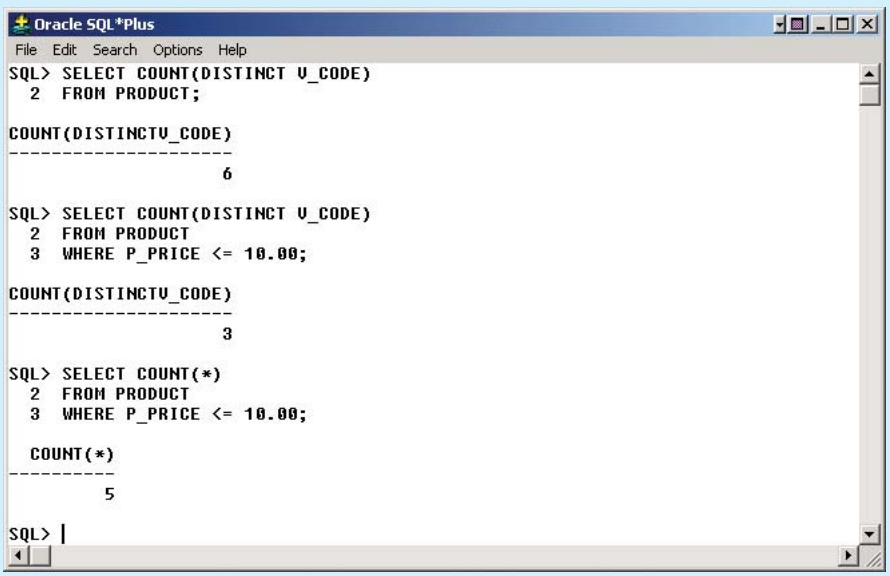
FUNCTION	OUTPUT
COUNT	The number of rows containing non-null values
MIN	The minimum attribute value encountered in a given column
MAX	The maximum attribute value encountered in a given column
SUM	The sum of all values for a given column
AVG	The arithmetic mean (average) for a specified column

To illustrate another standard SQL command format, most of the remaining input and output sequences are presented using the Oracle RDBMS.

COUNT

The **COUNT** function is used to tally the number of non-null values of an attribute. COUNT can be used in conjunction with the DISTINCT clause. For example, suppose that you want to find out how many different vendors are in the PRODUCT table. The answer, generated by the first SQL code set shown in Figure 7.21, is 6. The answer indicates that six different VENDOR codes are found in the PRODUCT table. (Note that the nulls are not counted as V_CODE values.)

FIGURE 7.21 COUNT function output examples



The aggregate functions can be combined with the SQL commands explored earlier. For example, the second SQL command set in Figure 7.21 supplies the answer to the question, "How many vendors referenced in the PRODUCT table have supplied products with prices that are less than or equal to $10?" The answer is three, indicating that three vendors referenced in the PRODUCT table have supplied products that meet the price specification.

The COUNT aggregate function uses one parameter within parentheses, generally a column name such as COUNT(V_CODE) or COUNT(P_CODE). The parameter may also be an expression such as COUNT(DISTINCT V_CODE) or COUNT(P_PRICE+10). Using that syntax, COUNT always returns the number of non-null values in the

given column. (Whether the column values are computed or show stored table row values is immaterial.) In contrast, the syntax COUNT(*) returns the number of total rows returned by the query, including the rows that contain nulls. In the example in Figure 7.21, SELECT COUNT(P_CODE) FROM PRODUCT and SELECT COUNT(*) FROM PRODUCT will yield the same answer because there are no null values in the P_CODE primary key column.

Note that the third SQL command set in Figure 7.21 uses the COUNT(*) command to answer the question, "How many rows in the PRODUCT table have a P_PRICE value less than or equal to $10?" The answer, five, indicates that five products have a listed price that meets the price specification. The COUNT(*) aggregate function is used to count rows in a query result set. In contrast, the COUNT(*column*) aggregate function counts the number of non-null values in a given column. For example, in Figure 7.20, the COUNT(*) function would return a value of 7 to indicate seven rows returned by the query. The COUNT(V_CODE) function would return a value of 6 to indicate the six non-null vendor code values.

NOTE

NOTE TO MS ACCESS USERS
MS Access does not support the use of COUNT with the DISTINCT clause. If you want to use such queries in MS Access, you must create subqueries with DISTINCT and NOT NULL clauses. For example, the equivalent MS Access queries for the first two queries shown in Figure 7.21 are:

```
SELECT      COUNT(*)
FROM        (SELECT DISTINCT V_CODE FROM PRODUCT WHERE V_CODE IS NOT NULL)
```

and

```
SELECT COUNT(*)
FROM (SELECT DISTINCT(V_CODE)
        FROM (SELECT V_CODE, P_PRICE FROM PRODUCT
                WHERE V_CODE IS NOT NULL AND P_PRICE<10))
```

Those two queries can be found in the Premium Website in the **Ch07_SaleCo** (Access) database. MS Access does add a trailer at the end of the query after you have executed it, but you can delete that trailer the next time you use the query.

MAX and MIN

The **MAX** and **MIN** functions help you find answers to problems such as the:

- Highest (maximum) price in the PRODUCT table.
- Lowest (minimum) price in the PRODUCT table.

The highest price, $256.99, is supplied by the first SQL command set in Figure 7.22. The second SQL command set shown in Figure 7.22 yields the minimum price of $4.99.

The third SQL command set in Figure 7.22 demonstrates that the numeric functions can be used in conjunction with more complex queries. However, you must remember that *the numeric functions yield only one value* based on all of the values found in the table: a single maximum value, a single minimum value, a single count, or a single average value. *It is easy to overlook this warning*. For example, examine the question, "Which product has the highest price?"

Although that query seems simple enough, the SQL command sequence:

```
SELECT      P_CODE, P_DESCRIPT, P_PRICE
FROM        PRODUCT
WHERE       P_PRICE = MAX(P_PRICE);
```

does not yield the expected results. This is because the use of MAX(P_PRICE) to the right side of a comparison operator is incorrect, thus producing an error message. The aggregate function MAX(*columnname*) can be used only

in the column list of a SELECT statement. Also, in a comparison that uses an equality symbol, you can use only a single value to the right of the equals sign.

To answer the question, therefore, you must compute the maximum price first, then compare it to each price returned by the query. To do that, you need a nested query. In this case, the nested query is composed of two parts:

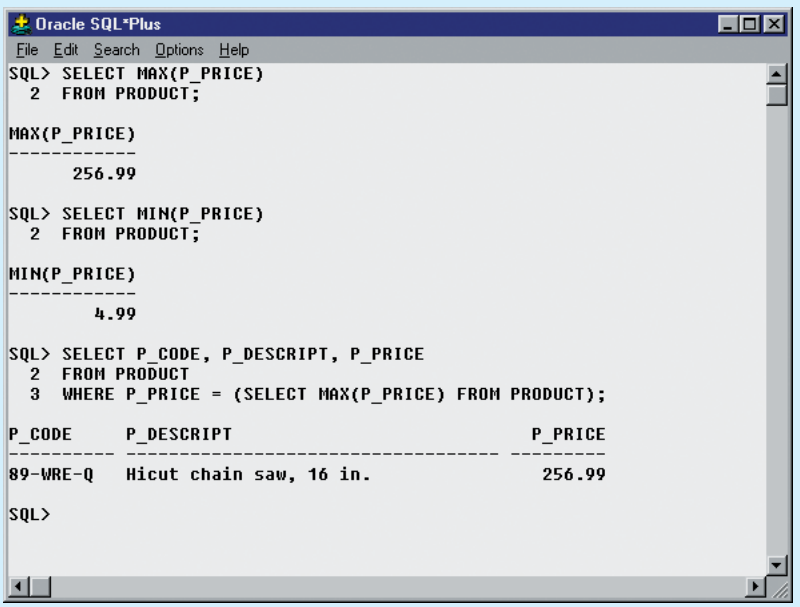
- The *inner query*, which is executed first.
- The *outer query*, which is executed last. (Remember that the outer query is always the first SQL command you encounter—in this case, SELECT.)

Using the following command sequence as an example, note that the inner query first finds the maximum price value, which is stored in memory. Because the outer query now has a value to which to compare each P_PRICE value, the query executes properly.

```
SELECT      P_CODE, P_DESCRIPT, P_PRICE
FROM        PRODUCT
WHERE       P_PRICE = (SELECT MAX(P_PRICE) FROM PRODUCT);
```

The execution of that nested query yields the correct answer, shown below the third (nested) SQL command set in Figure 7.22.

FIGURE 7.22 MAX and MIN output examples



The MAX and MIN aggregate functions can also be used with date columns. For example, to find out the product that has the oldest date, you would use MIN(P_INDATE). In the same manner, to find out the most recent product, you would use MAX(P_INDATE).

NOTE

You can use expressions anywhere a column name is expected. Suppose that you want to know what product has the highest inventory value. To find the answer, you can write the following query:

```
SELECT      *
FROM        PRODUCT
WHERE       P_QOH*P_PRICE = (SELECT MAX(P_QOH*P_PRICE) FROM PRODUCT);
```

SUM

The **SUM** function computes the total sum for any specified attribute, using whatever condition(s) you have imposed. For example, if you want to compute the total amount owed by your customers, you could use the following command:

```
SELECT      SUM(CUS_BALANCE) AS TOTBALANCE
FROM        CUSTOMER;
```

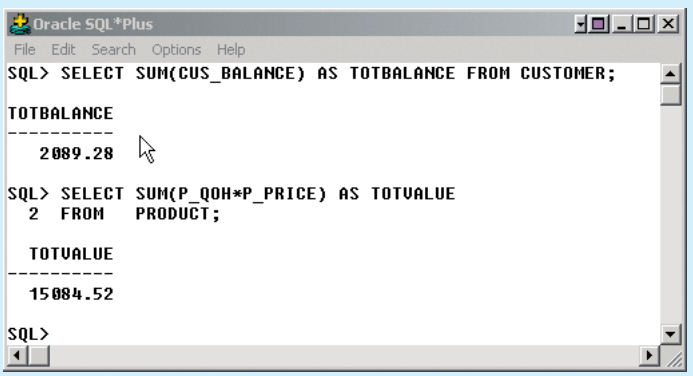
You could also compute the sum total of an expression. For example, if you want to find the total value of all items carried in inventory, you could use:

```
SELECT      SUM(P_QOH * P_PRICE) AS TOTVALUE
FROM        PRODUCT;
```

because the total value is the sum of the product of the quantity on hand and the price for all items. (See Figure 7.23.)

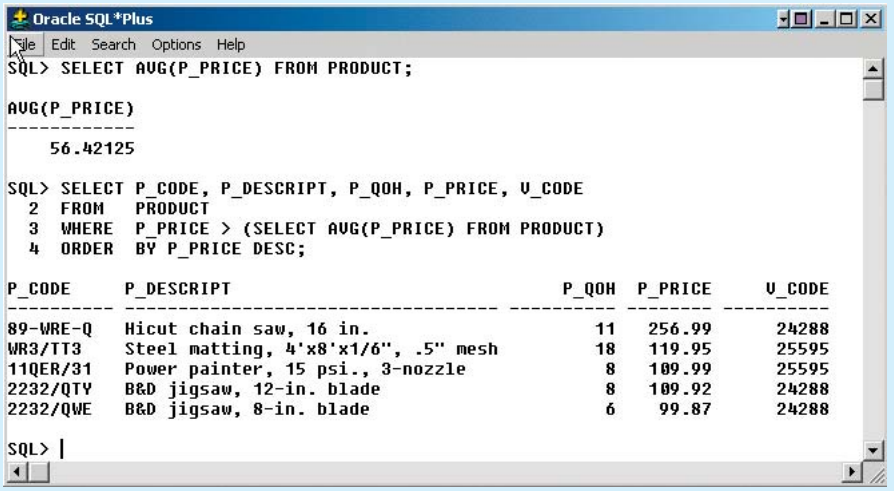
AVG

FIGURE 7.23 The total value of all items in the PRODUCT table



The **AVG** function format is similar to those of MIN and MAX and is subject to the same operating restrictions. The first SQL command set shown in Figure 7.24 shows how a simple average P_PRICE value can be generated to yield the computed average price of 56.42125. The second SQL command set in Figure 7.24 produces five output lines that describe products whose prices exceed the average product price. Note that the second query uses nested SQL commands and the ORDER BY clause examined earlier.

FIGURE 7.24 AVG function output examples



7.6.4 Grouping Data

Frequency distributions can be created quickly and easily using the **GROUP BY** clause within the SELECT statement. The syntax is:

SELECT *columnlist*
FROM *tablelist*
[WHERE *conditionlist*]
[GROUP BY *columnlist*]
[HAVING *conditionlist*]
[ORDER BY *columnlist* [ASC | DESC]] ;

The GROUP BY clause is generally used when you have attribute columns combined with aggregate functions in the SELECT statement. For example, to determine the minimum price for each sales code, use the first SQL command set shown in Figure 7.25.

The second SQL command set in Figure 7.25 generates the average price within each sales code. Note that the P_SALECODE nulls are included within the grouping.

The GROUP BY clause is valid only when used in conjunction with one of the SQL aggregate functions, such as COUNT, MIN, MAX, AVG, and SUM. For example, as shown in the first command set in Figure 7.26, if you try to group the output by using:

SELECT V_CODE, P_CODE, P_DESCRIPT, P_PRICE
FROM PRODUCT
GROUP BY V_CODE;

you generate a "not a GROUP BY expression" error. However, if you write the preceding SQL command sequence in conjunction with some aggregate function, the GROUP BY clause works properly. The second SQL command sequence in Figure 7.26 properly answers the question, "How many products are supplied by each vendor?" because it uses a COUNT aggregate function.

FIGURE 7.25 GROUP BY clause output examples

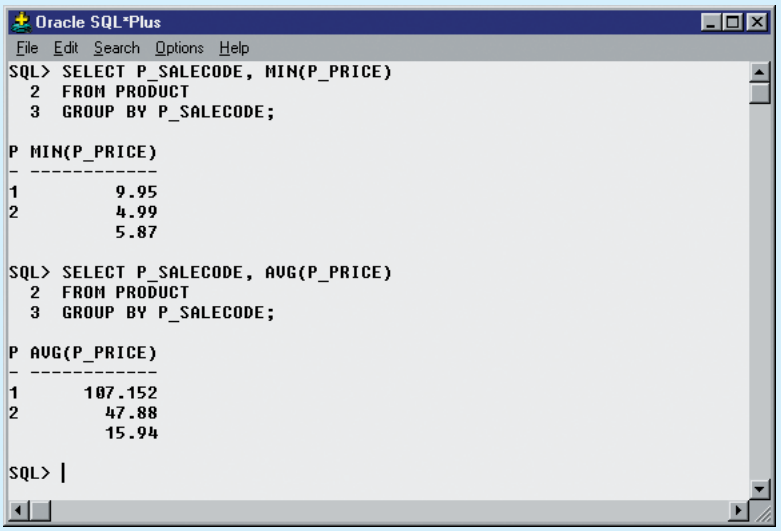
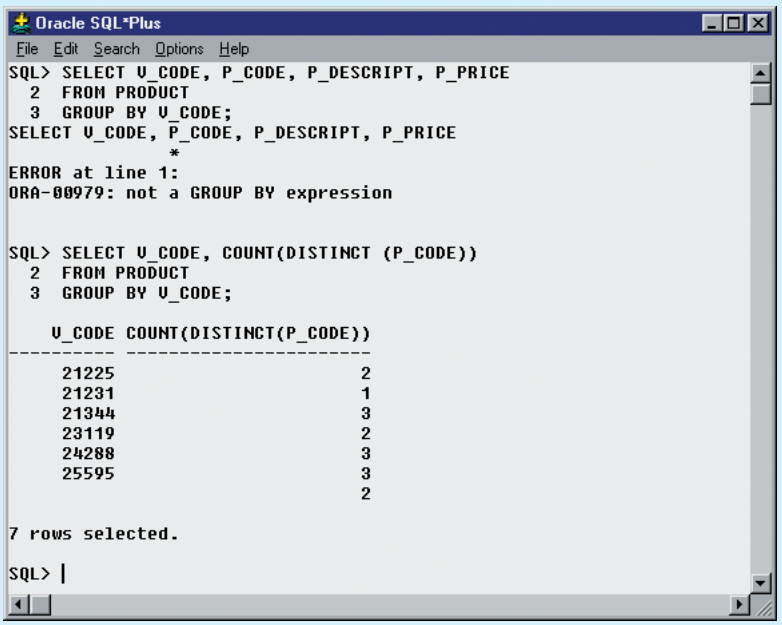


FIGURE 7.26 Incorrect and correct use of the GROUP BY clause



Note that the last output line in Figure 7.26 shows a null for the V_CODE, indicating that two products were not supplied by a vendor. Perhaps those products were produced in-house, or they might have been bought via a nonvendor channel, or the person making the data entry might have merely forgotten to enter a vendor code. (Remember that nulls can be the result of many things.)

NOTE

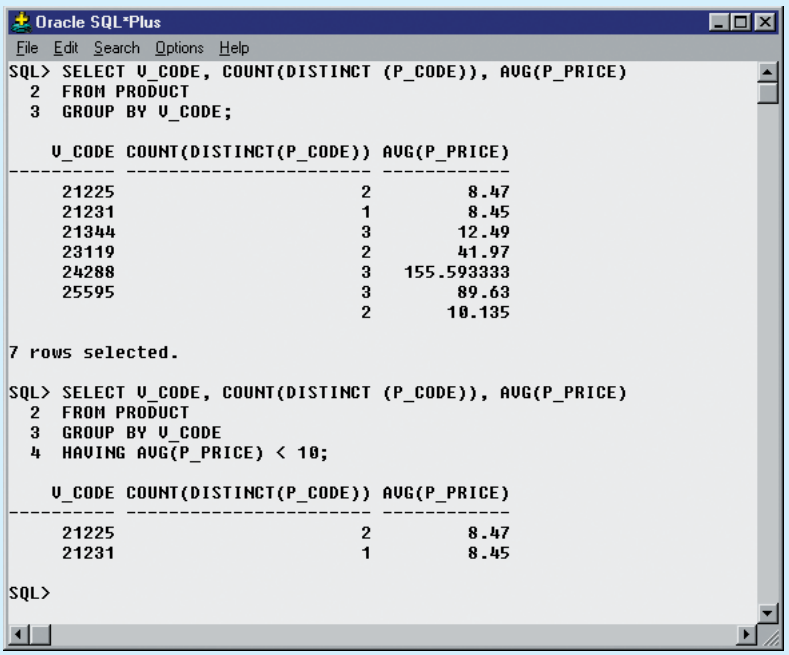
When using the GROUP BY clause with a SELECT statement:

- The SELECTs *columnlist* must include a combination of column names and aggregate functions.
- The GROUP BY clauses *columnlist* must include all nonaggregate function columns specified in the SELECTs *columnlist*. If required, you could also group by any aggregate function columns that appear in the SELECTs *columnlist*.
- The GROUP BY clause *columnlist* can include any columns from the tables in the FROM clause of the SELECT statement, even if they do not appear in the SELECTs *columnlist*.

The GROUP BY Feature's HAVING Clause

A particularly useful extension of the GROUP BY feature is the **HAVING** clause. The HAVING clause operates very much like the WHERE clause in the SELECT statement. However, the WHERE clause applies to columns and expressions for individual rows, while the HAVING clause is applied to the output of a GROUP BY operation. For example, suppose that you want to generate a listing of the number of products in the inventory supplied by each vendor. However, this time you want to limit the listing to products whose prices average below $10. The first part of that requirement is satisfied with the help of the GROUP BY clause, as illustrated in the first SQL command set in Figure 7.27. Note that the HAVING clause is used in conjunction with the GROUP BY clause in the second SQL command set in Figure 7.27 to generate the desired result.

FIGURE 7.27 An application of the HAVING clause



If you use the WHERE clause instead of the HAVING clause—the second SQL command set in Figure 7.27 will produce an error message.

You can also combine multiple clauses and aggregate functions. For example, consider the following SQL statement:

```
SELECT      V_CODE, SUM(P_QOH * P_PRICE) AS TOTCOST
FROM        PRODUCT
GROUP BY    V_CODE
HAVING      (SUM(P_QOH * P_PRICE) > 500)
ORDER BY    SUM(P_QOH * P_PRICE) DESC;
```

This statement will do the following:

- Aggregate the total cost of products grouped by V_CODE.
- Select only the rows having totals that exceed $500.
- List the results in descending order by the total cost.

Note the syntax used in the HAVING and ORDER BY clauses; in both cases, you must specify the column expression (formula) used in the SELECT statement's column list, rather than the column alias (TOTCOST). Some RDBMSs allow you to replace the column expression with the column alias, while others do not.

7.7 VIRTUAL TABLES: CREATING A VIEW

As you learned earlier, the output of a relational operator such as SELECT is another relation (or table). Suppose that at the end of every day, you would like to get a list of all products to reorder, that is, products with a quantity on hand that is less than or equal to the minimum quantity. Instead of typing the same query at the end of every day, wouldn't it be better to permanently save that query in the database? That's the function of a relational view. A **view** is a virtual table based on a SELECT query. The query can contain columns, computed columns, aliases, and aggregate functions from one or more tables. The tables on which the view is based are called **base tables**.

You can create a view by using the **CREATE VIEW** command:

CREATE VIEW *viewname* AS SELECT *query*

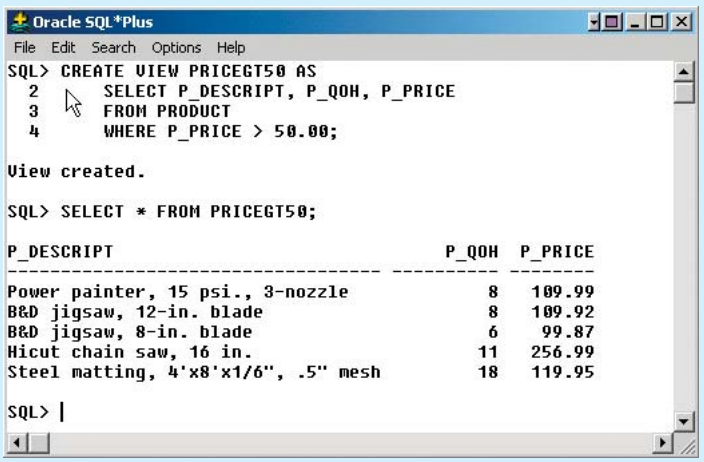
The CREATE VIEW statement is a data definition command that stores the subquery specification—the SELECT statement used to generate the virtual table—in the data dictionary.

The first SQL command set in Figure 7.28 shows the syntax used to create a view named PRICEGT50. This view contains only the designated three attributes (P_DESCRIPT, P_QOH, and P_PRICE) and only rows in which the price is over $50. The second SQL command sequence in Figure 7.28 shows the rows that make up the view.

A relational view has several special characteristics:

- You can use the name of a view anywhere a table name is expected in a SQL statement.
- Views are dynamically updated. That is, the view is re-created on demand each time it is invoked. Therefore, if new products are added (or deleted) to meet the criterion P_PRICE > 50.00, those new products will automatically appear (or disappear) in the PRICEGT50 view the next time the view is invoked.
- Views provide a level of security in the database because the view can restrict users to only specified columns and specified rows in a table. For example, if you have a company with hundreds of employees in several departments, you could give the secretary of each department a view of only certain attributes and for the employees that belong only to that secretary's department.

FIGURE 7.28 Creating a virtual table with the CREATE VIEW command



NOTE

NOTE TO MS ACCESS USERS
The CREATE VIEW command is not directly supported in MS Access. To create a view in MS Access, you just need to create a SQL query and then save it.

- Views may also be used as the basis for reports. For example, if you need a report that shows a summary of total product cost and quantity-on-hand statistics grouped by vendor, you could create a PROD_STATS view as:

```
CREATE VIEW PROD_STATS AS
SELECT      V_CODE, SUM(P_QOH*P_PRICE) AS TOTCOST,
            MAX(P_QOH) AS MAXQTY, MIN(P_QOH) AS MINQTY,
            AVG(P_QOH) AS AVGQTY
FROM        PRODUCT
GROUP BY    V_CODE;
```

In Chapter 8, you will learn more about views and, in particular, about updating data in base tables through views.

7.8 JOINING DATABASE TABLES

The ability to combine (join) tables on common attributes is perhaps the most important distinction between a relational database and other databases. A join is performed when data are retrieved from more than one table at a time. (If necessary, review the join definitions and examples in Chapter 3, The Relational Database Model.)

To join tables, you simply list the tables in the FROM clause of the SELECT statement. The DBMS will create the Cartesian product of every table in the FROM clause. (Review Chapter 3 to revisit these terms, if necessary.) However, to get the correct result—that is, a natural join—you must select only the rows in which the common attribute values match. To do this, use the WHERE clause to indicate the common attributes used to link the tables (this WHERE clause is sometimes referred to as the *join condition*).

The join condition is generally composed of an equality comparison between the foreign key and the primary key of related tables. For example, suppose that you want to join the two tables VENDOR and PRODUCT. Because V_CODE is the foreign key in the PRODUCT table and the primary key in the VENDOR table, the link is established on V_CODE. (See Table 7.9.)

TABLE 7.9 Creating Links Through Foreign Keys

TABLE	ATTRIBUTES TO BE SHOWN	LINKING ATTRIBUTE
PRODUCT	P_DESCRIPT, P_PRICE	V_CODE
VENDOR	V_COMPANY, V_PHONE	V_CODE

When the same attribute name appears in more than one of the joined tables, the source table of the attributes listed in the SELECT command sequence must be defined. To join the PRODUCT and VENDOR tables, you would use the following, which produces the output shown in Figure 7.29:

```
SELECT    P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      PRODUCT, VENDOR
WHERE     PRODUCT.V_CODE = VENDOR.V_CODE;
```

FIGURE 7.29 The results of a join

P_DESCRIPT	P_PRICE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE
Claw hammer	9.95	Bryson, Inc.	Smithson	615	223-3234
1.25-in. metal screw, 25	6.99	Bryson, Inc.	Smithson	615	223-3234
2.5-in. wd. screw, 50	8.45	D&E Supply	Singh	615	228-3245
7.25-in. pwr. saw blade	14.99	Gomez Bros.	Ortega	615	889-2546
9.00-in. pwr. saw blade	17.49	Gomez Bros.	Ortega	615	889-2546
Rat-tail file, 1/8-in. fine	4.99	Gomez Bros.	Ortega	615	889-2546
Hrd. cloth, 1/4-in., 2x50	39.95	Randsets Ltd.	Anderson	901	678-3998
Hrd. cloth, 1/2-in., 3x50	43.99	Randsets Ltd.	Anderson	901	678-3998
B&D jigsaw, 12-in. blade	109.92	ORDVA, Inc.	Hakford	615	898-1234
B&D jigsaw, 8-in. blade	99.87	ORDVA, Inc.	Hakford	615	898-1234
Hicut chain saw, 16 in.	256.99	ORDVA, Inc.	Hakford	615	898-1234
Power painter, 15 psi., 3-nozzle	109.99	Rubicon Systems	Orton	904	456-0092
B&D cordless drill, 1/2-in.	38.95	Rubicon Systems	Orton	904	456-0092
Steel matting, 4'x8'x1/6", .5" mesh	119.95	Rubicon Systems	Orton	904	456-0092

Your output might be presented in a different order because the SQL command produces a listing in which the order of the columns is not relevant. In fact, you are likely to get a different order of the same listing the next time you execute the command. However, you can generate a more predictable list by using an ORDER BY clause:

```
SELECT    PRODUCT.P_DESCRIPT, PRODUCT.P_PRICE, VENDOR.V_NAME, VENDOR.V_CONTACT,
          VENDOR.V_AREACODE, VENDOR.V_PHONE
FROM      PRODUCT, VENDOR
WHERE     PRODUCT.V_CODE = VENDOR.V_CODE
ORDER BY  PRODUCT.P_PRICE;
```

In that case, your listing will always be arranged from the lowest price to the highest price.

NOTE

Table names were used as prefixes in the preceding SQL command sequence. For example, PRODUCT.P_PRICE was used rather than P_PRICE. Most current-generation RDBMSs do not require table names to be used as prefixes unless the same attribute name occurs in several of the tables being joined. In that case, V_CODE is used as a foreign key in PRODUCT and as a primary key in VENDOR; therefore, you must use the table names as prefixes in the WHERE clause. In other words, you can write the previous query as:

```
SELECT    P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      PRODUCT, VENDOR WHERE PRODUCT.V_CODE = VENDOR.V_CODE
          ORDER BY P_PRICE;
```

Naturally, if an attribute name occurs in several places, its origin (table) must be specified. If you fail to provide such a specification, SQL will generate an error message to indicate that you have been ambiguous about the attribute's origin.

The preceding SQL command sequence joins a row in the PRODUCT table with a row in the VENDOR table in which the V_CODE values of these rows are the same, as indicated in the WHERE clause's condition. Because any vendor can deliver any number of ordered products, the PRODUCT table might contain multiple V_CODE entries for each V_CODE entry in the VENDOR table. In other words, each V_CODE in VENDOR can be matched with many V_CODE rows in PRODUCT.

If you do not specify the WHERE clause, the result will be the Cartesian product of PRODUCT and VENDOR. Because the PRODUCT table contains 16 rows and the VENDOR table contains 11 rows, the Cartesian product will produce a listing of (16 × 11) = 176 rows. (Each row in PRODUCT will be joined to each row in the VENDOR table.)

All of the SQL commands can be used on the joined tables. For example, the following command sequence is quite acceptable in SQL and produces the output shown in Figure 7.30:

```
SELECT    P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      PRODUCT, VENDOR
WHERE     PRODUCT.V_CODE = VENDOR.V_CODE
AND       P_INDATE > '15-Jan-2010';
```

FIGURE 7.30 An ordered and limited listing after a join

P_DESCRIPT	P_PRICE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE
1.25-in. metal screw, 25	6.99	Bryson, Inc.	Smithson	615	223-3234
2.5-in. wd. screw, 50	8.45	D&E Supply	Singh	615	228-3245
Claw hammer	9.95	Bryson, Inc.	Smithson	615	223-3234
B&D cordless drill, 1/2-in.	38.95	Rubicon Systems	Orton	904	456-0092
Steel matting, 4'x8'x1/6", .5" mesh	119.95	Rubicon Systems	Orton	904	456-0092
Hicut chain saw, 16 in.	256.99	ORDVA, Inc.	Hakford	615	898-1234

When joining three or more tables, you need to specify a join condition for each pair of tables. The number of join conditions will always be N-1, where N represents the number of tables listed in the FROM clause. For example, if you have three tables, you must have two join conditions; if you have five tables, you must have four join conditions; and so on.

Remember, the join condition will match the foreign key of a table to the primary key of the related table. For example, using Figure 7.1, if you want to list the customer last name, invoice number, invoice date, and product descriptions for all invoices for customer 10014, you must type the following:

```
SELECT    CUS_LNAME, INVOICE.INV_NUMBER, INV_DATE, P_DESCRIPT
FROM      CUSTOMER, INVOICE, LINE, PRODUCT
WHERE     CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
AND       INVOICE.INV_NUMBER = LINE.INV_NUMBER
AND       LINE.P_CODE = PRODUCT.P_CODE
AND       CUSTOMER.CUS_CODE = 10014
ORDER BY  INV_NUMBER;
```

Finally, be careful not to create circular join conditions. For example, if Table A is related to Table B, Table B is related to Table C, and Table C is also related to Table A, create only two join conditions: join A with B and B with C. Do not join C with A!

7.8.1 Joining Tables with an Alias

An alias may be used to identify the source table from which the data are taken. The aliases P and V are used to label the PRODUCT and VENDOR tables in the next command sequence. Any legal table name may be used as an alias. (Also notice that there are no table name prefixes because the attribute listing contains no duplicate names in the SELECT statement.)

```
SELECT    P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      PRODUCT P, VENDOR V
WHERE     P.V_CODE = V.V_CODE
ORDER BY  P_PRICE;
```

7.8.2 Recursive Joins

An alias is especially useful when a table must be joined to itself in a **recursive query**. For example, suppose that you are working with the EMP table shown in Figure 7.31.

FIGURE 7.31 The contents of the EMP table

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE	EMP_AREACODE	EMP_PHONE	EMP_MGR
100	Mr.	Kolmycz	George	D	15-Jun-42	15-Mar-85	615	324-5456	
101	Ms.	Lewis	Rhonda	G	19-Mar-65	25-Apr-86	615	324-4472	100
102	Mr.	Vandam	Rhett		14-Nov-58	20-Dec-90	901	675-8993	100
103	Ms.	Jones	Anne	M	16-Oct-74	28-Aug-94	615	898-3456	100
104	Mr.	Lange	John	P	08-Nov-71	20-Oct-94	901	504-4430	105
105	Mr.	Williams	Robert	D	14-Mar-75	08-Nov-98	615	890-3220	
106	Mrs.	Smith	Jeanine	K	12-Feb-68	05-Jan-89	615	324-7883	105
107	Mr.	Diante	Jorge	D	21-Aug-74	02-Jul-94	615	890-4567	105
108	Mr.	Wiesenbach	Paul	R	14-Feb-66	18-Nov-92	615	897-4358	
109	Mr.	Smith	George	K	18-Jun-61	14-Apr-89	901	504-3339	108
110	Mrs.	Genkazi	Leighla	W	19-May-70	01-Dec-90	901	569-0093	108
111	Mr.	Washington	Rupert	E	03-Jan-66	21-Jun-93	615	890-4925	105
112	Mr.	Johnson	Edward	E	14-May-61	01-Dec-83	615	898-4387	100
113	Ms.	Smythe	Melanie	P	15-Sep-70	11-May-99	615	324-9006	105
114	Ms.	Brandon	Marie	G	02-Nov-56	15-Nov-79	901	882-0845	108
115	Mrs.	Saranda	Hermine	R	25-Jul-72	23-Apr-93	615	324-5505	105
116	Mr.	Smith	George	A	08-Nov-65	10-Dec-88	615	890-2984	108

Using the data in the EMP table, you can generate a list of all employees with their managers' names by joining the EMP table to itself. In that case, you would also use aliases to differentiate the table from itself. The SQL command sequence would look like this:

```
SELECT      E.EMP_MGR, M.EMP_LNAME, E.EMP_NUM, E.EMP_LNAME
FROM        EMP E, EMP M
WHERE       E.EMP_MGR=M.EMP_NUM
ORDER BY    E.EMP_MGR;
```

The output of the preceding command sequence is shown in Figure 7.32.

FIGURE 7.32 Using an alias to join a table to itself

EMP_NUM	A.EMP_LNAME	EMP_MGR	B.EMP_LNAME
112	Johnson	100	Kolmycz
103	Jones	100	Kolmycz
102	Vandam	100	Kolmycz
101	Lewis	100	Kolmycz
115	Saranda	105	Williams
113	Smythe	105	Williams
111	Washington	105	Williams
107	Diante	105	Williams
106	Smith	105	Williams
104	Lange	105	Williams
116	Smith	108	Wiesenbach
114	Brandon	108	Wiesenbach
110	Genkazi	108	Wiesenbach
109	Smith	108	Wiesenbach

7.8.3 OUTER JOINS

Figure 7.29 showed the results of joining the PRODUCT and VENDOR tables. If you examine the output, note that 14 product rows are listed. Compare the output to the PRODUCT table in Figure 7.2, and note that two products are missing. Why? The reason is that there are two products with nulls in the V_CODE attribute. Because there is no matching null "value" in the VENDOR table's V_CODE attribute, the products do not show up in the final output based on the join. Also, note that in the VENDOR table in Figure 7.2, several vendors have no matching V_CODE in the PRODUCT table. To include those rows in the final join output, you must use an outer join.

NOTE

In MS Access, add AS to the previous SQL command sequence, for example:

```
SELECT      E.EMP_MGR,M.EMP_LNAME,E.EMP_NUM,E.EMP_LNAME
FROM        EMP AS E, EMP AS M
WHERE       E.EMP_MGR = M.EMP_NUM
ORDER BY    E.EMP_MGR;
```

There are two types of outer joins: left and right. (See Chapter 3.) Given the contents of the PRODUCT and VENDOR tables, the following left outer join will show all VENDOR rows and all matching PRODUCT rows:

```
SELECT      P_CODE, VENDOR.V_CODE, V_NAME
FROM        VENDOR LEFT JOIN PRODUCT
             ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

Figure 7.33 shows the output generated by the left outer join command in MS Access. Oracle yields the same result but shows the output in a different order.

The right outer join will join both tables and show all product rows with all matching vendor rows. The SQL command for the right outer join is:

```
SELECT      PRODUCT.P_CODE, VENDOR.V_CODE, V_NAME
FROM        VENDOR RIGHT JOIN PRODUCT
             ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

Figure 7.34 shows the output generated by the right outer join command sequence in MS Access. Again, Oracle yields the same result but shows the output in a different order.

In Chapter 8, you will learn more about joins and how to use the latest ANSI SQL standard syntax.

FIGURE 7.33 The left outer join results

P_CODE	V_CODE	V_NAME
23109-HB	21225	Bryson, Inc.
SM-18277	21225	Bryson, Inc.
	21226	SuperLoo, Inc.
SW-23116	21231	D&E Supply
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
54778-2T	21344	Gomez Bros.
	22567	Dome Supply
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
	24004	Brackman Bros.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
89-WRE-Q	24288	ORDVA, Inc.
	25443	B&K, Inc.
	25501	Damal Supplies
11QER/31	25595	Rubicon Systems
2238/QPD	25595	Rubicon Systems
WR3/TT3	25595	Rubicon Systems

FIGURE 7.34 The right outer join results

P_CODE	V_CODE	V_NAME
23114-AA		
PVC23DRT		
23109-HB	21225	Bryson, Inc.
SM-18277	21225	Bryson, Inc.
SW-23116	21231	D&E Supply
13-Q2/P2	21344	Gomez Bros.
14-Q1/L3	21344	Gomez Bros.
54778-2T	21344	Gomez Bros.
1546-QQ2	23119	Randsets Ltd.
1558-QW1	23119	Randsets Ltd.
2232/QTY	24288	ORDVA, Inc.
2232/QWE	24288	ORDVA, Inc.
89-WRE-Q	24288	ORDVA, Inc.
11QER/31	25595	Rubicon Systems
2238/QPD	25595	Rubicon Systems
WR3/TT3	25595	Rubicon Systems

ONLINE CONTENT

For a complete walk-through example of converting an ER model into a database structure and using SQL commands to create tables, see **Appendix D, Converting the ER Model into a Database Structure**, in the Premium Website for this book.

SUMMARY

- The SQL commands can be divided into two overall categories: data definition language (DDL) commands and data manipulation language (DML) commands.
- The ANSI standard data types are supported by all RDBMS vendors in different ways. The basic data types are NUMBER, INTEGER, CHAR, VARCHAR, and DATE.
- The basic data definition commands allow you to create tables, indexes, and views. Many SQL constraints can be used with columns. The commands are CREATE TABLE, CREATE INDEX, CREATE VIEW, ALTER TABLE, DROP TABLE, DROP VIEW, and DROP INDEX.
- DML commands allow you to add, modify, and delete rows from tables. The basic DML commands are SELECT, INSERT, UPDATE, DELETE, COMMIT, and ROLLBACK.
- The INSERT command is used to add new rows to tables. The UPDATE command is used to modify data values in existing rows of a table. The DELETE command is used to delete rows from tables. The COMMIT and ROLLBACK commands are used to permanently save or roll back changes made to the rows. Once you COMMIT the changes, you cannot undo them with a ROLLBACK command.
- The SELECT statement is the main data retrieval command in SQL. A SELECT statement has the following syntax:

```
SELECT      columnlist
FROM        tablelist
[WHERE      conditionlist ]
[GROUP BY   columnlist ]
[HAVING     conditionlist ]
[ORDER BY   columnlist [ASC | DESC] ] ;
```

- The column list represents one or more column names separated by commas. The column list may also include computed columns, aliases, and aggregate functions. A computed column is represented by an expression or formula (for example, P_PRICE * P_QOH). The FROM clause contains a list of table names or view names.
- The WHERE clause can be used with the SELECT, UPDATE, and DELETE statements to restrict the rows affected by the DDL command. The condition list represents one or more conditional expressions separated by logical operators (AND, OR, and NOT). The conditional expression can contain any comparison operators (=, >, <, >=, <=, and <>) as well as special operators (BETWEEN, IS NULL, LIKE, IN, and EXISTS).
- Aggregate functions (COUNT, MIN, MAX, and AVG) are special functions that perform arithmetic computations over a set of rows. The aggregate functions are usually used in conjunction with the GROUP BY clause to group the output of aggregate computations by one or more attributes. The HAVING clause is used to restrict the output of the GROUP BY clause by selecting only the aggregate rows that match a given condition.
- The ORDER BY clause is used to sort the output of a SELECT statement. The ORDER BY clause can sort by one or more columns and can use either ascending or descending order.
- You can join the output of multiple tables with the SELECT statement. The join operation is performed every time you specify two or more tables in the FROM clause and use a join condition in the WHERE clause to match the foreign key of one table to the primary key of the related table. If you do not specify a join condition, the DBMS will automatically perform a Cartesian product of the tables you specify in the FROM clause.
- The natural join uses the join condition to match only rows with equal values in the specified columns. You could also do a right outer join and left outer join to select the rows that have no matching values in the other related table.

KEY TERMS

alias, 246
ALTER TABLE, 253
AND, 248
authentication, 225
AVG, 265
base tables, 269
BETWEEN, 249
Boolean algebra, 248
cascading order sequence, 260
COMMIT, 238
COUNT, 262
CREATE INDEX, 235
CREATE TABLE, 229
CREATE VIEW, 269
DELETE, 241
DISTINCT, 261
DROP INDEX, 236
DROP TABLE, 259
EXISTS, 249
GROUP BY, 266
HAVING, 268
IN, 249
inner query, 242
INSERT, 237
IS NULL, 249
LIKE, 249
MAX, 263
MIN, 263
nested query, 242
NOT, 248
OR, 247
ORDER BY, 259
recursive query, 273
reserved words, 231
ROLLBACK, 240
rules of precedence, 247
schema, 225
SELECT, 238
subquery, 242
SUM, 265
UPDATE, 240
view, 269
wildcard character, 239

ONLINE CONTENT

Answers to selected Review Questions and Problems for this chapter are contained in the Premium Website for this book.

REVIEW QUESTIONS

1. In a SELECT query, what is the difference between a WHERE clause and a HAVING clause?
2. Explain why the following command would create an error and what changes could be made to fix the error.
   SELECT V_CODE, SUM(P_QOH) FROM PRODUCT;
3. What type of integrity is enforced when a primary key is declared?
4. Explain why it might be more appropriate to declare an attribute that contains only digits as a character data type instead of a numeric data type.
5. What is the difference between a column constraint and a table constraint?
6. What are "referential constraint actions"?
7. Rewrite the following WHERE clause without the use of the IN special operator.
   WHERE V_STATE IN ('TN', 'FL', 'GA')
8. Explain the difference between an ORDER BY clause and a GROUP BY clause.
9. Explain why the two following commands produce different results:
   SELECT DISTINCT COUNT (V_CODE) FROM PRODUCT;
   SELECT COUNT (DISTINCT V_CODE) FROM PRODUCT;
10. What is the difference between the COUNT aggregate function and the SUM aggregate function?
11. Explain why it would be preferable to use a DATE data type to store date data instead of a character data type.
12. What is the difference between an inner join and an outer join?

ONLINE CONTENT

Problems 1–25 are based on the **Ch07_ConstructCo** database located in the Premium Website. This database is stored in Microsoft Access format. If you use another DBMS such as Oracle, SQL Server, MySQL, or DB2, use its import utilities to import the Access database contents. The Premium Website provides Oracle and SQL script files.

PROBLEMS

The **Ch07_ConstructCo** database stores data for a consulting company that tracks all charges to projects. The charges are based on the hours each employee works on each project. The structure and contents of the **Ch07_ConstructCo** database are shown in Figure P7.1.

FIGURE P7.1 The Ch07_ConstructCo database

Database name: Ch07_ConstructCo

Relational diagram

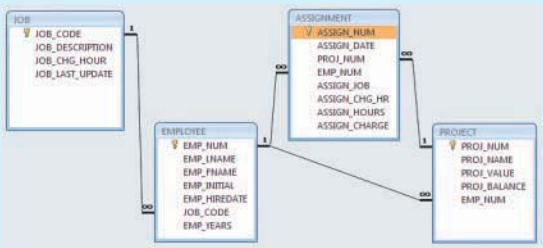


Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	EMP_YEARS
101	News	John	G	08-Nov-00	502	8
102	Senior	David	H	12-Jul-89	501	19
103	Arbough	June	E	01-Dec-96	503	12
104	Ramoras	Anne	K	15-Nov-87	501	21
105	Johnson	Alice	K	01-Feb-93	502	16
106	Smithfield	William		22-Jun-04	500	4
107	Alonzo	Maria	D	10-Oct-93	500	15
108	Washington	Ralph	B	22-Aug-91	501	17
109	Smith	Larry	W	18-Jul-97	501	11
110	Olenko	Gerald	A	11-Dec-95	505	13
111	Wabash	Geoff	B	04-Apr-91	506	17
112	Smithson	Darlene	M	23-Oct-94	507	14
113	Joenbrood	Delbert	K	15-Nov-96	508	12
114	Jones	Annelise		20-Aug-93	508	15
115	Bawangi	Travis	B	25-Jan-92	501	17
116	Pratt	Gerald	L	06-Mar-97	510	12
117	Williamson	Angie	H	19-Jun-96	509	12
118	Frommer	James	J	04-Jan-05	510	4

Table name: JOB

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR	JOB_LAST_UPDATE
500	Programmer	35.75	20-Nov-09
501	Systems Analyst	96.75	20-Nov-09
502	Database Designer	125.00	24-Mar-10
503	Electrical Engineer	84.50	20-Nov-09
504	Mechanical Engineer	67.90	20-Nov-09
505	Civil Engineer	55.78	20-Nov-09
506	Clerical Support	26.87	20-Nov-09
507	DSS Analyst	45.95	20-Nov-09
508	Applications Designer	48.10	24-Mar-10
509	Bio Technician	34.55	20-Nov-09
510	General Support	18.36	20-Nov-09

Table name: PROJECT

PROJ_NUM	PROJ_NAME	PROJ_VALUE	PROJ_BALANCE	EMP_NUM
15	Evergreen	1453500.00	1002350.00	103
18	Amber Wave	3500500.00	2110346.00	108
22	Rolling Tide	805000.00	500345.20	102
25	Starflight	2650500.00	2309880.00	107

Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_JOB	ASSIGN_CHG_HR	ASSIGN_HOURS	ASSIGN_CHARGE
1001	22-Mar-10	18	103	503	84.50	3.5	295.75
1002	22-Mar-10	22	117	509	34.55	4.2	145.11
1003	22-Mar-10	18	117	509	34.55	2.0	69.10
1004	22-Mar-10	18	103	503	84.50	5.9	498.55
1005	22-Mar-10	25	108	501	96.75	2.2	212.85
1006	22-Mar-10	22	104	501	96.75	4.2	406.35
1007	22-Mar-10	25	113	508	50.75	3.8	192.85
1008	22-Mar-10	18	103	503	84.50	0.9	76.05
1009	23-Mar-10	15	115	501	96.75	5.6	541.80
1010	23-Mar-10	15	117	509	34.55	2.4	82.92
1011	23-Mar-10	25	105	502	105.00	4.3	451.50
1012	23-Mar-10	18	108	501	96.75	3.4	328.95
1013	23-Mar-10	25	115	501	96.75	2.0	193.50
1014	23-Mar-10	22	104	501	96.75	2.8	270.90
1015	23-Mar-10	15	103	503	84.50	6.1	515.45
1016	23-Mar-10	22	105	502	105.00	4.7	493.50
1017	23-Mar-10	18	117	509	34.55	3.8	131.29
1018	23-Mar-10	25	117	509	34.55	2.2	76.01
1019	24-Mar-10	25	104	501	110.50	4.9	541.45
1020	24-Mar-10	15	101	502	125.00	3.1	387.50
1021	24-Mar-10	22	108	501	110.50	2.7	298.35
1022	24-Mar-10	22	115	501	110.50	4.9	541.45
1023	24-Mar-10	22	105	502	125.00	3.5	437.50
1024	24-Mar-10	15	103	503	84.50	3.3	278.85
1025	24-Mar-10	18	117	509	34.55	4.2	145.11

Note that the ASSIGNMENT table in Figure P7.1 stores the JOB_CHG_HOUR values as an attribute (ASSIGN_CHG_HR) to maintain historical accuracy of the data. The JOB_CHG_HOUR values are likely to change over time. In fact, a JOB_CHG_HOUR change will be reflected in the ASSIGNMENT table. And, naturally, the employee primary job assignment might change, so the ASSIGN_JOB is also stored. Because those attributes are required to maintain the historical accuracy of the data, they are *not* redundant.

Given the structure and contents of the **Ch07_ConstructCo** database shown in Figure P7.1, use SQL commands to answer Problems 1–25.

1. Write the SQL code that will create the table structure for a table named EMP_1. This table is a subset of the EMPLOYEE table. The basic EMP_1 table structure is summarized in the following table. (Note that the JOB_CODE is the FK to JOB.)

ATTRIBUTE (FIELD) NAME	DATA DECLARATION
EMP_NUM	CHAR(3)
EMP_LNAME	VARCHAR(15)
EMP_FNAME	VARCHAR(15)
EMP_INITIAL	CHAR(1)
EMP_HIREDATE	DATE
JOB_CODE	CHAR(3)

2. Having created the table structure in Problem 1, write the SQL code to enter the first two rows for the table shown in Figure P7.2.

FIGURE P7.2 **The contents of the EMP_1 table**

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-96	500
104	Ramoras	Anne	K	15-Nov-87	501
105	Johnson	Alice	K	01-Feb-93	502
106	Smithfield	William		22-Jun-04	500
107	Alonzo	Maria	D	10-Oct-93	500
108	Washington	Ralph	B	22-Aug-91	501
109	Smith	Larry	W	18-Jul-97	501

3. Assuming that the data shown in the EMP_1 table have been entered, write the SQL code that will list all attributes for a job code of 502.
4. Write the SQL code that will save the changes made to the EMP_1 table.
5. Write the SQL code to change the job code to 501 for the person whose employee number (EMP_NUM) is 107. After you have completed the task, examine the results, and then reset the job code to its original value.
6. Write the SQL code to delete the row for the person named William Smithfield, who was hired on June 22, 2004, and whose job code classification is 500. (*Hint*: Use logical operators to include all of the information given in this problem.)
7. Write the SQL code that will restore the data to its original status; that is, the table should contain the data that existed before you made the changes in Problems 5 and 6.

8. Write the SQL code to create a copy of EMP_1, naming the copy EMP_2. Then write the SQL code that will add the attributes EMP_PCT and PROJ_NUM to its structure. The EMP_PCT is the bonus percentage to be paid to each employee. The new attribute characteristics are:

   EMP_PCTNUMBER(4,2)

   PROJ_NUMCHAR(3)

   (*Note*: If your SQL implementation allows it, you may use DECIMAL(4,2) rather than NUMBER(4,2).)

9. Write the SQL code to change the EMP_PCT value to 3.85 for the person whose employee number (EMP_NUM) is 103. Next, write the SQL command sequences to change the EMP_PCT values as shown in Figure P7.9.

FIGURE P7.9 The EMP_2 table after the modifications

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	EMP_PCT	PROJ_NUM
101	News	John	G	08-Nov-00	502	5.00	
102	Senior	David	H	12-Jul-89	501	8.00	
103	Arbough	June	E	01-Dec-96	500	3.85	
104	Ramoras	Anne	K	15-Nov-87	501	10.00	
105	Johnson	Alice	K	01-Feb-93	502	5.00	
106	Smithfield	William		22-Jun-04	500	6.20	
107	Alonzo	Maria	D	10-Oct-93	500	5.15	
108	Washington	Ralph	B	22-Aug-91	501	10.00	
109	Smith	Larry	W	18-Jul-97	501	2.00	

10. Using a single command sequence, write the SQL code that will change the project number (PROJ_NUM) to 18 for all employees whose job classification (JOB_CODE) is 500.
11. Using a single command sequence, write the SQL code that will change the project number (PROJ_NUM) to 25 for all employees whose job classification (JOB_CODE) is 502 or higher. When you finish Problems 10 and 11, the EMP_2 table will contain the data shown in Figure P7.11. (You may assume that the table has been saved again at this point.)

FIGURE P7.11 The EMP_2 table contents after the modifications

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	EMP_PCT	PROJ_NUM
101	News	John	G	08-Nov-00	502	5.00	25
102	Senior	David	H	12-Jul-89	501	8.00	
103	Arbough	June	E	01-Dec-96	500	3.85	18
104	Ramoras	Anne	K	15-Nov-87	501	10.00	
105	Johnson	Alice	K	01-Feb-93	502	5.00	25
106	Smithfield	William		22-Jun-04	500	6.20	18
107	Alonzo	Maria	D	10-Oct-93	500	5.15	18
108	Washington	Ralph	B	22-Aug-91	501	10.00	
109	Smith	Larry	W	18-Jul-97	501	2.00	

12. Write the SQL code that will change the PROJ_NUM to 14 for those employees who were hired before January 1, 1994 and whose job code is at least 501. (You may assume that the table will be restored to its condition preceding this question.)

13. Write the two SQL command sequences required to:
    a. Create a temporary table named TEMP_1 whose structure is composed of the EMP_2 attributes EMP_NUM and EMP_PCT.
    b. Copy the matching EMP_2 values into the TEMP_1 table.
14. Write the SQL command that will delete the newly created TEMP_1 table from the database.
15. Write the SQL code required to list all employees whose last names start with *Smith*. In other words, the rows for both Smith and Smithfield should be included in the listing. Assume case sensitivity.
16. Using the EMPLOYEE, JOB, and PROJECT tables in the **Ch07_ConstructCo** database (see Figure P7.1), write the SQL code that will produce the results shown in Figure P7.16.

FIGURE P7.16 The query results for Question 16

PROJ_NAME	PROJ_VALUE	PROJ_BALANCE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
Rolling Tide	805000.00	500345.20	Senior	David	H	501	Systems Analyst	96.75
Evergreen	1453500.00	1002350.00	Arbough	June	E	500	Programmer	35.75
Starflight	2650500.00	2309880.00	Alonzo	Maria	D	500	Programmer	35.75
Amber Wave	3500500.00	2110346.00	Washington	Ralph	B	501	Systems Analyst	96.75

17. Write the SQL code that will produce a virtual table named REP_1. The virtual table should contain the same information that was shown in Problem 16.
18. Write the SQL code to find the average bonus percentage in the EMP_2 table you created in Problem 8.
19. Write the SQL code that will produce a listing for the data in the EMP_2 table in ascending order by the bonus percentage.
20. Write the SQL code that will list only the distinct project numbers found in the EMP_2 table.
21. Write the SQL code to calculate the ASSIGN_CHARGE values in the ASSIGNMENT table in the **Ch07_ConstructCo** database. (See Figure P7.1.) Note that ASSIGN_CHARGE is a derived attribute that is calculated by multiplying ASSIGN_CHG_HR by ASSIGN_HOURS.
22. Using the data in the ASSIGNMENT table, write the SQL code that will yield the total number of hours worked for each employee and the total charges stemming from those hours worked. The results of running that query are shown in Figure P7.22.

FIGURE P7.22 Total hours and charges by employee

EMP_NUM	EMP_LNAME	SumOfASSIGN_HOURS	SumOfASSIGN_CHARGE
101	News	3.1	387.50
103	Arbough	19.7	1664.65
104	Ramoras	11.9	1218.70
105	Johnson	12.5	1382.50
108	Washington	8.3	840.15
113	Joenbrood	3.8	192.85
115	Bawangi	12.5	1276.75
117	Williamson	18.8	649.54

23. Write a query to produce the total number of hours and charges for each of the projects represented in the ASSIGNMENT table. The output is shown in Figure P7.23.

FIGURE P7.23 Total hour and charges by project

PROJ_NUM	SumOfASSIGN_HOURS	SumOfASSIGN_CHARGE
15	20.5	1806.52
18	23.7	1544.80
22	27.0	2593.16
25	19.4	1668.16

FIGURE P7.24 Total hours and charges, all employees

SumOfSumOfASSIGN_HOURS	SumOfSumOfASSIGN_CHARGE
90.6	7612.64

24. Write the SQL code to generate the total hours worked and the total charges made by all employees. The results are shown in Figure P7.24. (*Hint:* This is a nested query. If you use Microsoft Access, you can generate the result by using the query output shown in Figure P7.22 as the basis for the query that will produce the output shown in Figure P7.24.)
25. Write the SQL code to generate the total hours worked and the total charges made to all projects. The results should be the same as those shown in Figure P7.24. (*Hint:* This is a nested query. If you use Microsoft Access, you can generate the result by using the query output shown in Figure P7.23 as the basis for this query.)

    The structure and contents of the **Ch07_SaleCo** database are shown in Figure P7.26. Use this database to answer the following problems. Save each query as QXX, where XX is the problem number.

ONLINE CONTENT

Problems 26–43 are based on the **Ch07_SaleCo** database located in the Premium Website. This database is stored in Microsoft Access format. If you use another DBMS such as Oracle, SQL Server, MySQL, or DB2, use its import utilities to import the Access database contents. The Premium Website provides Oracle and SQL script files.

26. Write a query to count the number of invoices.
27. Write a query to count the number of customers with a customer balance over $500.
28. Generate a listing of all purchases made by the customers, using the output shown in Figure P7.28 as your guide. (*Hint:* Use the ORDER BY clause to order the resulting rows shown in Figure P7.28.)

FIGURE P7.26 The Ch07_SaleCo database

Relational diagram

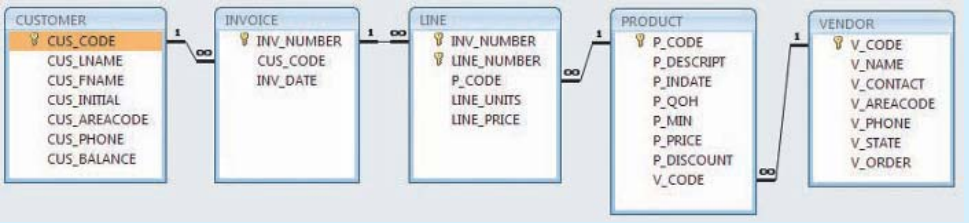


Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
10010	Ramas	Alfred	A	615	844-2573	0.00
10011	Dunne	Leona	K	713	894-1238	0.00
10012	Smith	Kathy	W	615	894-2285	345.86
10013	Olowski	Paul	F	615	894-2180	536.75
10014	Orlando	Myron		615	222-1672	0.00
10015	O'Brian	Amy	B	713	442-3381	0.00
10016	Brown	James	G	615	297-1228	221.19
10017	Williams	George		615	290-2556	768.93
10018	Farriss	Anne	G	713	382-7185	216.55
10019	Smith	Olette	K	615	297-3809	0.00

Table name: VENDOR

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-0092	FL	Y

Table name: INVOICE

INV_NUMBER	CUS_CODE	INV_DATE
1001	10014	16-Jan-10
1002	10011	16-Jan-10
1003	10012	16-Jan-10
1004	10011	17-Jan-10
1005	10018	17-Jan-10
1006	10014	17-Jan-10
1007	10015	17-Jan-10
1008	10011	17-Jan-10

Table name: LINE

INV_NUMBER	LINE_NUMBER	P_CODE	LINE_UNITS	LINE_PRICE
1001	1	13-Q2/P2	1	14.99
1001	2	23109-HB	1	9.95
1002	1	54778-2T	2	4.99
1003	1	2238/QPD	1	38.95
1003	2	1546-QQ2	1	39.95
1003	3	13-Q2/P2	5	14.99
1004	1	54778-2T	3	4.99
1004	2	23109-HB	2	9.95
1005	1	PVC23DRT	12	5.87
1006	1	SM-18277	3	6.99
1006	2	2232/QTY	1	109.92
1006	3	23109-HB	1	9.95
1006	4	89-WRE-Q	1	256.99
1007	1	13-Q2/P2	2	14.99
1007	2	54778-2T	1	4.99
1008	1	PVC23DRT	5	5.87
1008	2	WR3/TT3	3	119.95
1008	3	23109-HB	1	9.95

Table name: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-09	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-09	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-09	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-10	15	8	39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-10	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-09	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-09	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-10	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-10	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Jan-10	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-09	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-10	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-10	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-10	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-10	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-10	18	5	119.95	0.10	25595

FIGURE P7.28 List of customer purchases

CUS_CODE	INV_NUMBER	INV_DATE	P_DESCRIPT	LINE_UNITS	LINE_PRICE
10011	1002	16-Jan-10	Rat-tail file, 1/8-in. fine	2	4.99
10011	1004	17-Jan-10	Claw hammer	2	9.95
10011	1004	17-Jan-10	Rat-tail file, 1/8-in. fine	3	4.99
10011	1008	17-Jan-10	Claw hammer	1	9.95
10011	1008	17-Jan-10	PVC pipe, 3.5-in., 8-ft	5	5.87
10011	1008	17-Jan-10	Steel matting, 4'x8'x1/6", .5" mesh	3	119.95
10012	1003	16-Jan-10	7.25-in. pwr. saw blade	5	14.99
10012	1003	16-Jan-10	B&D cordless drill, 1/2-in.	1	38.95
10012	1003	16-Jan-10	Hrd. cloth, 1/4-in., 2x50	1	39.95
10014	1001	16-Jan-10	7.25-in. pwr. saw blade	1	14.99
10014	1001	16-Jan-10	Claw hammer	1	9.95
10014	1006	17-Jan-10	1.25-in. metal screw, 25	3	6.99
10014	1006	17-Jan-10	B&D jigsaw, 12-in. blade	1	109.92
10014	1006	17-Jan-10	Claw hammer	1	9.95
10014	1006	17-Jan-10	Hicut chain saw, 16 in.	1	256.99
10015	1007	17-Jan-10	7.25-in. pwr. saw blade	2	14.99
10015	1007	17-Jan-10	Rat-tail file, 1/8-in. fine	1	4.99
10018	1005	17-Jan-10	PVC pipe, 3.5-in., 8-ft	12	5.87

29. Using the output shown in Figure P7.29 as your guide, generate a list of customer purchases, including the subtotals for each of the invoice line numbers. (*Hint:* Modify the query format used to produce the list of customer purchases in Problem 28, delete the INV_DATE column, and add the derived (computed) attribute LINE_UNITS * LINE_PRICE to calculate the subtotals.)

FIGURE P7.29 Summary of customer purchases with subtotals

CUS_CODE	INV_NUMBER	P_DESCRIPT	Units Bought	Unit Price	Subtotal
10011	1002	Rat-tail file, 1/8-in. fine	2	4.99	9.98
10011	1004	Claw hammer	2	9.95	19.90
10011	1004	Rat-tail file, 1/8-in. fine	3	4.99	14.97
10011	1008	Claw hammer	1	9.95	9.95
10011	1008	PVC pipe, 3.5-in., 8-ft	5	5.87	29.35
10011	1008	Steel matting, 4'x8'x1/6", .5" mesh	3	119.95	359.85
10012	1003	7.25-in. pwr. saw blade	5	14.99	74.95
10012	1003	B&D cordless drill, 1/2-in.	1	38.95	38.95
10012	1003	Hrd. cloth, 1/4-in., 2x50	1	39.95	39.95
10014	1001	7.25-in. pwr. saw blade	1	14.99	14.99
10014	1001	Claw hammer	1	9.95	9.95
10014	1006	1.25-in. metal screw, 25	3	6.99	20.97
10014	1006	B&D jigsaw, 12-in. blade	1	109.92	109.92
10014	1006	Claw hammer	1	9.95	9.95
10014	1006	Hicut chain saw, 16 in.	1	256.99	256.99
10015	1007	7.25-in. pwr. saw blade	2	14.99	29.98
10015	1007	Rat-tail file, 1/8-in. fine	1	4.99	4.99
10018	1005	PVC pipe, 3.5-in., 8-ft	12	5.87	70.44

30. Modify the query used in Problem 29 to produce the summary shown in Figure P7.30.

FIGURE P7.30 Customer purchase summary

CUS_CODE	CUS_BALANCE	Total Purchases
10011	0.00	444.00
10012	345.86	153.85
10014	0.00	422.77
10015	0.00	34.97
10018	216.55	70.44

31. Modify the query in Problem 30 to include the number of individual product purchases made by each customer. (In other words, if the customer's invoice is based on three products, one per LINE_NUMBER, you count three product purchases. Note that in the original invoice data, customer 10011 generated three invoices, which contained a total of six lines, each representing a product purchase.) Your output values must match those shown in Figure P7.31.

32. Use a query to compute the average purchase amount per product made by each customer. (*Hint:* Use the results of Problem 31 as the basis for this query.) Your output values must match those shown in Figure P7.32. Note that the average purchase amount is equal to the total purchases divided by the number of purchases.

FIGURE P7.31 Customer total purchase amounts and number of purchases

CUS_CODE	CUS_BALANCE	Total Purchases	Number of Purchases
10011	0.00	444.00	6
10012	345.86	153.85	3
10014	0.00	422.77	6
10015	0.00	34.97	2
10018	216.55	70.44	1

33. Create a query to produce the total purchase per invoice, generating the results shown in Figure P7.33. The invoice total is the sum of the product purchases in the LINE that corresponds to the INVOICE.

FIGURE P7.32 Average purchase amount by customer

CUS_CODE	CUS_BALANCE	Total Purchases	Number of Purchases	Average Purchase Amount
10011	0.00	444.00	6	74.00
10012	345.86	153.85	3	51.28
10014	0.00	422.77	6	70.46
10015	0.00	34.97	2	17.48
10018	216.55	70.44	1	70.44

34. Use a query to show the invoices and invoice totals as shown in Figure P7.34. (*Hint:* Group by the CUS_CODE.)

FIGURE P7.33 Invoice totals

INV_NUMBER	Invoice Total
1001	24.94
1002	9.98
1003	153.85
1004	34.87
1005	70.44
1006	397.83
1007	34.97
1008	399.15

FIGURE P7.34 Invoice totals by customer

CUS_CODE	INV_NUMBER	Invoice Total
10011	1002	9.98
10011	1004	34.87
10011	1008	399.15
10012	1003	153.85
10014	1001	24.94
10014	1006	397.83
10015	1007	34.97
10018	1005	70.44

35. Write a query to produce the number of invoices and the total purchase amounts by customer, using the output shown in Figure P7.35 as your guide. (Compare this summary to the results shown in Problem 34.)
36. Using the query results in Problem 35 as your basis, write a query to generate the total number of invoices, the invoice total for all of the invoices, the smallest invoice amount, the largest invoice amount, and the average of all of the invoices. (*Hint:* Check the figure output in Problem 35.) Your output must match Figure P7.36.

FIGURE P7.35 Number of invoices and total purchase amounts by customer

CUS_CODE	Number of Invoices	Total Customer Purchases
10011	3	444.00
10012	1	153.85
10014	2	422.77
10015	1	34.97
10018	1	70.44

FIGURE P7.36 Number of invoices, invoice totals, minimum, maximum, and average sales

Total Invoices	Total Sales	Minimum Sale	Largest Sale	Average Sale
8	1126.03	34.97	444.00	225.21

37. List the balance characteristics of the customers who have made purchases during the current invoice cycle—that is, for the customers who appear in the INVOICE table. The results of this query are shown in Figure P7.37.

FIGURE P7.37 Balances for customers who made purchases

CUS_CODE	CUS_BALANCE
10011	0.00
10012	345.86
10014	0.00
10015	0.00
10018	216.55

FIGURE P7.38 Balance summary of customers who made purchases

Minimum Balance	Maximum Balance	Average Balance
0	345.86	112.48

38. Using the results of the query created in Problem 37, provide a summary of the customer balance characteristics as shown in Figure P7.38.
39. Create a query to find the customer balance characteristics for all customers, including the total of the outstanding balances. The results of this query are shown in Figure P7.39.
40. Find the listing of customers who did not make purchases during the invoicing period. Your output must match the output shown in Figure P7.40.

FIGURE P7.39 Balance summary for all customers

Total Balances	Minimum Balanace	Maximum Balance	Average Balance
2089.28	0.00	768.93	208.93

FIGURE P7.40 Balances of customers who did not make purchases

CUS_CODE	CUS_BALANCE
10010	0.00
10013	536.75
10016	221.19
10017	768.93
10019	0.00

41. Find the customer balance summary for all customers who have not made purchases during the current invoicing period. The results are shown in Figure P7.41.

FIGURE P7.41 Summary of customer balances for customers who did not make purchases

Total Balances	Minimum Balanace	Maximum Balance	Average Balance
1526.87	0.00	768.93	305.37

42. Create a query to produce the summary of the value of products currently in inventory. Note that the value of each product is produced by the multiplication of the units currently in inventory and the unit price. Use the ORDER BY clause to match the order shown in Figure P7.42.
43. Using the results of the query created in Problem 42, find the total value of the product inventory. The results are shown in Figure P7.43.

FIGURE P7.42 Value of products currently in inventory

P_DESCRIPT	P_QOH	P_PRICE	Subtotal
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

FIGURE P7.43 Total value of all products in inventory

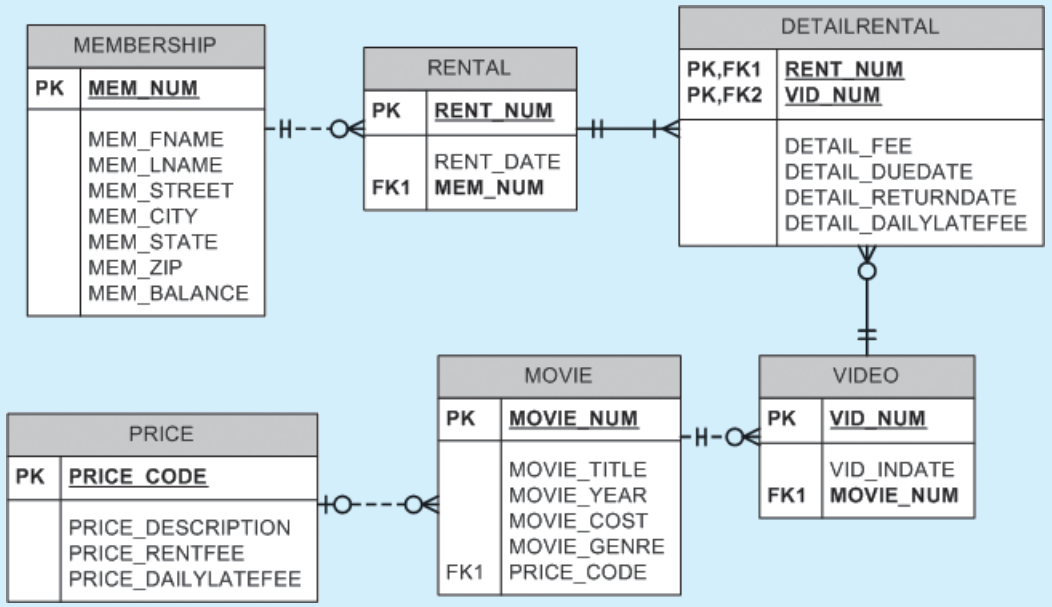
Total Value of Inventory
15084.52

CASES

TinyVideo is a small movie rental company with a single store. TinyVideo needs a database system to track the rental of movies to its members. TinyVideo can own several copies (VIDEO) of each movie (MOVIE). For example, the store may have 10 copies of the movie "Twist in the Wind." "Twist in the Wind" would be one MOVIE, and each copy would be a VIDEO. A rental transaction (RENTAL) involves one or more videos being rented to a member (MEMBERSHIP). A video can be rented many times over its lifetime; therefore, there is a M:N relationship between RENTAL and VIDEO. DETAILRENTAL is the bridge table to resolve this relationship. The complete ERD is provided in Figure P7.44.

44. Write the SQL code to create the table structures for the entities shown in Figure P7.44. The structures should contain the attributes specified in the ERD. Use data types that are appropriate for the data that will need to be stored in each attribute. Enforce primary key and foreign key constraints as indicated by the ERD.

FIGURE P7.44 The Ch07_MovieCo ERD



45. The following tables provide a very small portion of the data that will be kept in the database. This data needs to be inserted into the database for testing purposes. Write the INSERT commands necessary to place the following data in the tables that were created in Problem 1.

MEMBERSHIP

MEM_NUM	MEM_FNAME	MEM_LNAME	MEM_STREET	MEM_CITY	MEM_STATE	MEM_ZIP	MEM_BALANCE
102	Tami	Dawson	2632 Takli Circle	Norene	TN	37136	11
103	Curt	Knight	4025 Cornell Court	Flatgap	KY	41219	6
104	Jamal	Melendez	788 East 145th Avenue	Quebeck	TN	38579	0
105	Iva	Mcclain	6045 Musket Ball Circle	Summit	KY	42783	15
106	Miranda	Parks	4469 Maxwell Place	Germantown	TN	38183	0
107	Rosario	Elliott	7578 Danner Avenue	Columbia	TN	38402	5
108	Mattie	Guy	4390 Evergreen Street	Lily	KY	40740	0
109	Clint	Ochoa	1711 Elm Street	Greeneville	TN	37745	10
110	Lewis	Rosales	4524 Southwind Circle	Counce	TN	38326	0
111	Stacy	Mann	2789 East Cook Avenue	Murfreesboro	TN	37132	8
112	Luis	Trujillo	7267 Melvin Avenue	Heiskell	TN	37754	3
113	Minnie	Gonzales	6430 Vasili Drive	Williston	TN	38076	0

RENTAL		
RENT_NUM	RENT_DATE	MEM_NUM
1001	01-MAR-09	103
1002	01-MAR-09	105
1003	02-MAR-09	102
1004	02-MAR-09	110
1005	02-MAR-09	111
1006	02-MAR-09	107
1007	02-MAR-09	104
1008	03-MAR-09	105
1009	03-MAR-09	111

DETAILRENTAL					
RENT_NUM	VID_NUM	DETAIL_FEE	DETAIL_DUEDATE	DETAIL_RETURNDATE	DETAIL_DAILYLATEFEE
1001	34342	2	04-MAR-09	02-MAR-09	1
1001	61353	2	04-MAR-09	03-MAR-09	1
1002	59237	3.5	04-MAR-09	04-MAR-09	3
1003	54325	3.5	04-MAR-09	09-MAR-09	3
1003	61369	2	06-MAR-09	09-MAR-09	1
1003	61388	0	06-MAR-09	09-MAR-09	1
1004	44392	3.5	05-MAR-09	07-MAR-09	3
1004	34367	3.5	05-MAR-09	07-MAR-09	3
1004	34341	2	07-MAR-09	07-MAR-09	1
1005	34342	2	07-MAR-09	05-MAR-09	1
1005	44397	3.5	05-MAR-09	05-MAR-09	3
1006	34366	3.5	05-MAR-09	04-MAR-09	3
1006	61367	2	07-MAR-09		1
1007	34368	3.5	05-MAR-09		3
1008	34369	3.5	05-MAR-09	05-MAR-09	3
1009	54324	3.5	05-MAR-09		3
1001	34366	3.5	04-MAR-09	02-MAR-09	3

VIDEO		
VID_NUM	VID_INDATE	MOVIE_NUM
54321	18-JUN-08	1234
54324	18-JUN-08	1234
54325	18-JUN-08	1234
34341	22-JAN-07	1235
34342	22-JAN-07	1235
34366	02-MAR-09	1236
34367	02-MAR-09	1236
34368	02-MAR-09	1236
34369	02-MAR-09	1236
44392	21-OCT-08	1237
44397	21-OCT-08	1237
59237	14-FEB-09	1237
61388	25-JAN-07	1239
61353	28-JAN-06	1245
61354	28-JAN-06	1245
61367	30-JUL-08	1246
61369	30-JUL-08	1246

MOVIE					
MOVIE_NUM	MOVIE_NAME	MOVIE_YEAR	MOVIE_COST	MOVIE_GENRE	PRICE_CODE
1234	The Cesar Family Christmas	2007	39.95	FAMILY	2
1235	Smokey Mountain Wildlife	2004	59.95	ACTION	1
1236	Richard Goodhope	2008	59.95	DRAMA	2
1237	Beatnik Fever	2007	29.95	COMEDY	2
1238	Constant Companion	2008	89.95	DRAMA	2
1239	Where Hope Dies	1998	25.49	DRAMA	3
1245	Time to Burn	2005	45.49	ACTION	1
1246	What He Doesn't Know	2006	58.29	COMEDY	1

PRICE			
PRICE_CODE	PRICE_DESCRIPTION	PRICE_RENTFEE	PRICE_DAILYLATEFEE
1	Standard	2	1
2	New Release	3.5	3
3	Discount	1.5	1
4	Weekly Special	1	.5

For Questions 46–77, use the tables that were created in Problem 44 and the data that was loaded into those tables in Problem 45.

46. Write the SQL command to save the rows inserted in Problem 45.
47. Write the SQL command to change the movie year for movie number 1245 to 2006.
48. Write the SQL command to change the price code for all Action movies to price code 3.
49. Write a single SQL command to increase all price rental fee values by $0.50.
50. Write the SQL command to save the changes made to the PRICE and MOVIE tables in Problems 45–49.

51. Write a query to display the movie title, movie year, and movie genre for all movies. (The results are shown in Figure P7.51.)
52. Write a query to display the movie year, movie title, and movie cost sorted by movie year in descending order. (The results are shown in Figure P7.52.)

FIGURE P7.51 All movies

Movie_Title	Movie_Year	Movie_Genre
The Cesar Family C	2007	FAMILY
Smokey Mountain V	2004	ACTION
Richard Goodhope	2008	DRAMA
Beatnik Fever	2007	COMEDY
Constant Companio	2008	DRAMA
Where Hope Dies	1998	DRAMA
Time to Burn	2006	ACTION
What He Doesn't Kr	2006	COMEDY

FIGURE P7.52 Movies by year

Movie_Year	Movie_Title	Movie_Cost
2008	Constant Companio	89.95
2008	Richard Goodhope	59.95
2007	Beatnik Fever	29.95
2007	The Cesar Family C	39.95
2006	What He Doesn't Kr	58.29
2006	Time to Burn	45.49
2004	Smokey Mountain V	59.95
1998	Where Hope Dies	25.49

53. Write a query to display the movie title, movie year, and movie genre for all movies sorted by movie genre in ascending order, then sorted by movie year in descending order within genre. (The results are shown in Figure P7.53.)
54. Write a query to display the movie number, movie title, and price code for all movies with a title that starts with the letter "R." (The results are shown in Figure P7.54.)

FIGURE P7.53 Movies with multicolumn sort

Movie_Title	Movie_Year	Movie_Genre
Time to Burn	2006	ACTION
Smokey Mountain V	2004	ACTION
Beatnik Fever	2007	COMEDY
What He Doesn't Kr	2006	COMEDY
Constant Companio	2008	DRAMA
Richard Goodhope	2008	DRAMA
Where Hope Dies	1998	DRAMA
The Cesar Family C	2007	FAMILY

FIGURE P7.54 Movies starting with R

Movie_Num	Movie_Title	Price_Code
1236	Richard Goodhope	2

55. Write a query to display the movie title, movie year, and movie cost for all movies that contain the word "hope" anywhere in the title. Sort the results in ascending order by title. (The results are shown in figure P7.55.)
56. Write a query to display the movie title, movie year, and movie genre for all action movies. (The results are shown in Figure P7.56.)

FIGURE P7.55 Movies with "Hope" in the title

Movie_Title	Movie_Year	Movie_Cost
Richard Goodhope	2008	59.95
Where Hope Dies	1998	25.49

FIGURE P7.56 Action movies

Movie_Title	Movie_Year	Movie_Genre
Smokey Mountain V	2004	ACTION
Time to Burn	2006	ACTION

57. Write a query to display the movie number, movie title, and movie cost for all movies with a cost greater than $40. (The results are shown in Figure P7.57.)

58. Write a query to display the movie number, movie title, movie cost, and movie genre for all movies that are either action or comedy movies and that have a cost that is less than $50. Sort the results in ascending order by genre. (The results are shown in Figure P7.58.)

FIGURE P7.57 Movies costing less than $40

Movie_Num	Movie_Title	Movie_Cost
1235	Smokey Mountain Wildlife	59.95
1236	Richard Goodhope	59.95
1238	Constant Companion	89.95
1245	Time to Burn	45.49
1246	What He Doesn't Know	58.29

FIGURE P7.58 Action or comedy movies less than $50

Movie_Num	Movie_Title	Movie_Cost	Movie_Genre
1245	Time to Burn	45.49	ACTION
1235	Smokey Mountain Wildlife	59.95	ACTION
1246	What He Doesn't Know	58.29	COMEDY
1237	Beatnik Fever	29.95	COMEDY
1239	Where Hope Dies	25.49	DRAMA
1234	The Cesar Family Christmas	39.95	FAMILY

59. Write a query to display the movie number, and movie description for all movies where the movie description is a combination of the movie title, movie year, and movie genre with the movie year enclosed in parentheses. (The results are shown in Figure P7.59.)
60. Write a query to display the movie genre and the number of movies in each genre. (The results are shown in Figure P7.60.)

FIGURE P7.59 Movies with concatenated descriptions

Movie_Num	Movie Description
1234	The Cesar Family Christmas (2007) FAMILY
1235	Smokey Mountain Wildlife (2004) ACTION
1236	Richard Goodhope (2008) DRAMA
1237	Beatnik Fever (2007) COMEDY
1238	Constant Companion (2008) DRAMA
1239	Where Hope Dies (1998) DRAMA
1245	Time to Burn (2006) ACTION
1246	What He Doesn't Know (2006) COMEDY

FIGURE P7.60 Number of movies in genre

Movie_Genre	Number of Movies
ACTION	2
COMEDY	2
DRAMA	3
FAMILY	1

61. Write a query to display the average cost of all of the movies. (The results are shown in Figure P7.61.)
62. Write a query to display the movie genre and average cost of movies in each genre. (The results are shown in Figure P7.62.)

FIGURE P7.61 Average movie cost

Average Movie Cost
51.1275

FIGURE P7.62 Average cost by genre

Movie_Genre	Average Cost
ACTION	52.72
COMEDY	44.12
DRAMA	58.46
FAMILY	39.95

63. Write a query to display the movie title, movie genre, price description, and price rental fee for all movies with a price code. (The results are shown in Figure P7.63.)
64. Write a query to display the movie genre and average price rental fee for movies in each genre that have a price. (The results are shown in Figure P7.64.)

FIGURE P7.63 Rental fees for movies

Movie_Title	Movie_Genre	Price_Description	Price_RentFee
What He Doesn't Know	COMEDY	Standard	2.5
The Cesar Family Christmas	FAMILY	New Release	4
Richard Goodhope	DRAMA	New Release	4
Beatnik Fever	COMEDY	New Release	4
Smokey Mountain Wildlife	ACTION	Discount	2
Where Hope Dies	DRAMA	Discount	2
Time to Burn	ACTION	Discount	2

FIGURE P7.64 Average rental fee by genre

Movie_Genre	Average Rental Fee
ACTION	2
COMEDY	3.25
DRAMA	3
FAMILY	4

65. Write a query to display the movie title, movie year, and the movie cost divided by the price rental fee for each movie that has a price to determine the number of rentals it will take to break even on the purchase of the movie. (The results are shown in Figure P7.65.)
66. Write a query to display the movie title and movie year for all movies that have a price code. (The results are shown in Figure P7.66.)

FIGURE P7.65 Breakeven rentals

Movie_Title	Movie_Year	Breakeven Rentals
What He Doesn't Kr	2006	23.32
The Cesar Family C	2007	9.99
Richard Goodhope	2008	14.99
Beatnik Fever	2007	7.49
Smokey Mountain V	2004	29.98
Where Hope Dies	1998	12.75
Time to Burn	2006	22.75

FIGURE P7.66 Movies with a price

Movie_Title	Movie_Year
The Cesar Family Christmas	2007
Smokey Mountain Wildlife	2004
Richard Goodhope	2008
Beatnik Fever	2007
Where Hope Dies	1998
Time to Burn	2006
What He Doesn't Know	2006

67. Write a query to display the movie title, movie year, and movie cost for all movies that have a cost between $44.99 and $49.99. (The results are shown in Figure P7.67.)
68. Write a query to display the movie title, movie year, price description, and price rental fee for all movies that are in the genres family, comedy, or drama. (The results are shown in Figure P7.68.)

FIGURE P7.67 Movies costs within a range

Movie_Title	Movie_Year	Movie_Cost
Time to Burn	2006	45.49

FIGURE P7.68 Movies within specific genres

Movie_Title	Movie_Year	Price_Description	Price_RentFee	Movie_Genre
The Cesar Family Christmas	2007	New Release	4	FAMILY
Richard Goodhope	2008	New Release	4	DRAMA
Beatnik Fever	2007	New Release	4	COMEDY
Where Hope Dies	1998	Discount	2	DRAMA
What He Doesn't Know	2006	Standard	2.5	COMEDY

69. Write a query to display the movie number, movie title, and movie year for all movies that do not have a video. (The results are shown in Figure P7.69.)
70. Write a query to display the membership number, first name, last name, and balance of the memberships that have a rental. (The results are shown in Figure P7.70.)

FIGURE P7.69 Movies without videos

Movie_Num	Movie_Title	Movie_Year
1238	Constant Companion	2008

FIGURE P7.70 Balances of memberships with rentals

Mem_Num	Mem_FName	Mem_LName	Mem_Balance
102	Tami	Dawson	11
103	Curt	Knight	6
104	Jamal	Melendez	0
105	Iva	McClain	15
107	Rosario	Elliott	5
110	Lewis	Rosales	0
111	Stacy	Mann	8

71. Write a query to display the minimum balance, maximum balance, and average balance for memberships that have a rental. (The results are shown in Figure P7.71.)

72. Write a query to display the membership name (concatenate the first name and last name with a space between them into a single column), membership address (concatenate the street, city, state, and zip codes into a single column with spaces. (The results are shown in Figure P7.72.)

FIGURE P7.71 Minimum, maximum, and average balances

Minimum Balance	Maximum Balance	Average Balance
0	15	6.43

FIGURE P7.72 Concatenated membership data

Membership Name	Membership Address
Tami Dawson	2632 Takli Circle, Norene, TN 37136
Curt Knight	4025 Cornell Court, Flatgap, KY 41219
Jamal Melendez	788 East 145th Avenue, Quebeck, TN 38579
Iva McClain	6045 Musket Ball Circle, Summit, KY 42783
Miranda Parks	4469 Maxwell Place, Germantown, TN 38183
Rosario Elliott	7578 Danner Avenue, Columbia, TN 38402
Mattie Guy	4390 Evergreen Street, Lily, KY 40740
Clint Ochoa	1711 Elm Street, Greenville, TN 37745
Lewis Rosales	4524 SouthWind Circle, Counce, TN 38326
Stacy Mann	2789 East Cook Avenue, Murfreesboro, TN 37132
Luis Trujillo	7267 Melvin Avenue, Heiskell, TN 37754
Minnie Gonzales	6430 Vasili Drive, Williston, TN 38076

73. Write a query to display the rental number, rental date, video number, movie title, due date, and return date for all videos that were returned after the due date. Sort the results by rental number and movie title. (The results are shown in Figure P7.73.)

FIGURE P7.73 Late video returns

Rent_Num	Rent_Date	Vid_Num	Movie_Title	Detail_DueDate	Detail_ReturnDate
1003	02-Mar-09	54325	The Cesar Family Christmas	04-Mar-09	09-Mar-09
1003	02-Mar-09	61369	What He Doesn't Know	06-Mar-09	09-Mar-09
1003	02-Mar-09	61388	Where Hope Dies	06-Mar-09	09-Mar-09
1004	02-Mar-09	44392	Beatnik Fever	05-Mar-09	07-Mar-09
1004	02-Mar-09	34367	Richard Goodhope	05-Mar-09	07-Mar-09

74. Write a query to display the rental number, rental date, video number, movie title, due date, return date, detail fee, and number of days past the due date that the video was returned for each video that was returned after the due date. Sort the results by rental number and movie title. (The results are shown in Figure P7.74.)

FIGURE P7.74 Number of days late

Rent_Num	Rent_Date	Vid_Num	Movie_Title	Detail_DueDate	Detail_ReturnDate	Days Past Due
1003	02-Mar-09	54325	The Cesar Family Christmas	04-Mar-09	09-Mar-09	5
1003	02-Mar-09	61369	What He Doesn't Know	06-Mar-09	09-Mar-09	3
1003	02-Mar-09	61388	Where Hope Dies	06-Mar-09	09-Mar-09	3
1004	02-Mar-09	44392	Beatnik Fever	05-Mar-09	07-Mar-09	2
1004	02-Mar-09	34367	Richard Goodhope	05-Mar-09	07-Mar-09	2

75. Write a query to display the rental number, rental date, movie title, and detail fee for each movie that was returned on or before the due date. (The results are shown in Figure P7.75.)

FIGURE P7.75 Actual rental fees charged

Rent_Num	Rent_Date	Movie_Title	Detail_Fee
1001	01-Mar-09	Smokey Mountain Wildlife	2
1001	01-Mar-09	Time to Burn	2
1002	01-Mar-09	Beatnik Fever	3.5
1004	02-Mar-09	Smokey Mountain Wildlife	2
1005	02-Mar-09	Smokey Mountain Wildlife	2
1005	02-Mar-09	Beatnik Fever	3.5
1006	02-Mar-09	Richard Goodhope	3.5
1008	03-Mar-09	Richard Goodhope	3.5
1001	01-Mar-09	Richard Goodhope	3.5

76. Write a query to display the membership number, last name, first name, and total rental fees earned from that membership. (The results are shown in Figure P7.76.) The total rental fee is the sum of all of the detail fees (without the late fees) from all movies that the membership has rented.

FIGURE P7.76 Total rental fees paid by membership

Mem_Num	Mem_LName	Mem_FName	Rental Fee Revenue
102	Dawson	Tami	5.5
103	Knight	Curt	7.5
104	Melendez	Jamal	3.5
105	McClain	Iva	7
107	Elliott	Rosario	5.5
110	Rosales	Lewis	9
111	Mann	Stacy	9

77. Write a query to display the movie number, movie genre, average movie cost of movies in that genre, movie cost of that individual movie, and the percentage difference between the average movie cost and the individual movie cost. (The results are shown in Figure P7.77.) (*Note:* The percentage difference is calculated as the cost of the individual movie minus the average cost of movies in that genre, divided by the average cost of movies in that genre multiplied by 100. For example, if the average cost of movies in the "family" genre is $25, if a given family movie cost $26, then the calculation would be ((26 – 25) / 25 * 100), which would work out to be 4.00%. This indicates that this movie costs 4% more than the average family movie.)

FIGURE P7.77 Movie differences from genre average

Movie_Num	Movie_Genre	Average Cost	Movie_Cost	Percent Difference
1234	FAMILY	39.95	39.95	0.00
1235	ACTION	52.72	59.95	13.71
1236	DRAMA	58.46	59.95	2.54
1237	COMEDY	44.12	29.95	-32.12
1238	DRAMA	58.46	89.95	53.86
1239	DRAMA	58.46	25.49	-56.40
1245	ACTION	52.72	45.49	-13.71
1246	COMEDY	44.12	58.29	32.12