CHAPTER

2

Creating and Modifying Database Tables

Objectives

After completing this chapter, you should be able to:

- ♦ Use Structured Query Language (SQL) commands to create, modify, and drop database tables
- ♦ Explain Oracle10*g* user schemas
- ♦ Define Oracle10*g* database tables
- ♦ Create database tables using SQL*Plus
- ♦ Debug Oracle10*g* SQL commands and use online help resources available through the Oracle Technology Network (OTN)
- ♦ View information about your database tables using Oracle10*g* data dictionary views
- ♦ Modify and delete database tables using SQL*Plus

With Oracle10*g* you use Structured Query Language (SQL) for data entry, retrieval, and manipulation. SQL is the standard query language for relational databases. In this chapter, you learn how to use SQL*Plus, Oracle's command-line SQL utility to issue SQL commands. Because the first step in creating the database is to make tables, this chapter focuses on the SQL commands required to create and modify tables.

INTRODUCTION TO SQL

Users interact with relational databases using high-level query languages. Query languages have commands that contain standard English words such as CREATE, ALTER, INSERT, UPDATE, and DELETE. The standard query language for relational databases is **Structured Query Language (SQL)**. Basic SQL consists of about 30 commands that enable users to create database objects and manipulate and view data. The American National Standards Institute (ANSI) has published standards for SQL. The most recent version was published in 1999, and is called SQL-99. Most relational database vendors do not yet fully comply with SQL-99, but almost all relational database applications support the SQL ANSI standard published in 1992, which is called SQL-92. This book uses SQL commands that comply with the SQL-92 standard.

Most database vendors add extensions to the standard SQL commands to make their databases more powerful or easier to use. These extensions are fairly similar across different platforms, and once you become proficient with SQL on one DBMS platform, it is fairly easy to move to other platforms.

SQL commands fall into two basic categories:

- **Data definition language (DDL)** commands—Used to create new database objects (such as user accounts and tables) and modify or delete existing objects. When you execute a DDL command, the command immediately changes the database, so you do not need to save the change explicitly.
- **Data manipulation language (DML)** commands—Used to insert, update, delete, and view database data. When you execute a DML command, you must explicitly save the command to make the new data values visible to other database users.

In this chapter, you work with DDL commands to create new database tables. Usually, only database administrators execute DDL commands. Sometimes you execute DDL operations using utilities that automatically generate the underlying SQL commands. However, to create the applications in the tutorials and case projects in this book, you must first create the underlying database tables in your Oracle10*g* database. Furthermore, you must understand the underlying structure of the database tables. To gain this understanding, you must be familiar with the syntax and use of the SQL DDL commands that create the tables. In this chapter, you use the DDL commands to create, modify, and drop Oracle10*g* database tables. You type DDL commands into a text editor, and you execute the commands in SQL*Plus, the Oracle10*g* command-line SQL environment.

In this book, all SQL command words, which are known as **reserved words**, appear in all uppercase letters. All user-supplied variable names appear in lowercase letters, although Oracle10*g* displays the user-supplied values in uppercase letters when it executes the commands. Although SQL*Plus commands are not case sensitive, these conventions make SQL commands and the examples in this book easier to interpret. When the book references a table or column outside a command, the table or column name appears in all uppercase letters. When the book provides the general syntax for a command, user-supplied values appear in lowercase italic type. Optional parameters or optional phrases within commands appear in square brackets. For parameters that can have one value from a set of values, the possible values are separated by the bar (|) character.

ORACLE10*g* USER ACCOUNTS

Before you begin working with an Oracle10*g* database, you need to understand how Oracle10*g* manages database user accounts. When you create a new table using a personal database such as Access, you start the database application and create a new database. Access saves the database file in your workstation's file system. You are usually the only person who uses this database file. In contrast, when a database administrator creates a new Oracle10*g* database, the database server stores the database files, and all of the database's users share these files. To keep each user's tables and data separate and secure, the DBA creates a **user account** for each user, which he or she identifies using a unique username and password. Each user can then create database tables and other data objects that reside in his or her area of the database, which is called a **user schema**. The data objects within a user schema are called **database objects** or **schema objects**. A user's schema contains all of the objects that the user creates and stores in the database. Schema objects include all of the user's database tables. Schema objects also include views, which are logical tables that are based on a query, and stored programs.

To connect to an Oracle10*g* database and create and manage database objects in your user schema, you must enter a username and a password.

NOTE

If you are using an Oracle10*g* Personal Edition database, the directions for creating the username and password are in the installation instructions. If you are using an Oracle10*g* Enterprise Edition client/server database, your instructor will create your username and password.

TIP

As a security measure, Oracle does not automatically provide privileges when a new user account is created. In fact, even with a valid user account, you must still be explicitly granted the privilege to connect to the database.

When you create a new database object in your user schema, you are the owner of this object. As the object owner, you have privileges to perform all possible actions on the object; for example, you can modify a database table's structure, delete the table, and insert, update, and view table data.

DEFINING ORACLE10*g* DATABASE TABLES

Tables are the primary data objects in a relational database. When you create a new Oracle10*g* database table, you specify the table name, the name of each data column, and the data type and size of each data column. You can also define **constraints**, which are restrictions on the data values that a column can store. Examples of constraints include whether the column is a primary key, whether it is a foreign key, whether a column allows NULL values, and whether the column can only contain certain values, such as FR, SO, JR, or SR for a column that specifies student class values.

Table names and column names must follow the **Oracle naming standard**, which is a series of rules that Oracle Corporation has established for naming all database objects. This Oracle naming standard states that objects must be from one to 30 characters long, can contain letters, numbers, and the special symbols ($),(_), and (#), and must begin with a character. Examples of legal Oracle10*g* database object names are STUDENT_TABLE, PRICE$, or COURSE_ID#. Examples of illegal Oracle10*g* database object names are STUDENT TABLE (which contains a blank space), STUDENT-TABLE (which contains a hyphen), or #COURSE_ID (which does not begin with a character). The Oracle naming standard is referenced throughout the book.

You use the CREATE TABLE SQL command to create a new table. The general syntax for the CREATE TABLE command is:

```
CREATE TABLE tablename
(columnname1 data_type,
columnname2 data_type, ...)
```

In this syntax, *tablename* represents the name of the new table and must follow the Oracle naming standard. Following the CREATE TABLE clause, you list the name of each database column, followed by its data type. The data type identifies the type of data that the column stores, such as numbers, characters, or dates. (The next section describes Oracle10*g* data types in detail.) Database column names must also follow the Oracle naming standard. You enclose the list of columns in parentheses, and separate each column specification with a comma.

The CREATE TABLE command is an example of a SQL DDL command. (Recall that you use DDL commands to create and modify database objects.) With a CREATE TABLE command, the DBMS creates the database table as soon as the command executes, and you do not need to use any other commands to explicitly save the table.

ORACLE10*g* DATA TYPES

Notice in the syntax for creating a table that you specify a data type for each column. The **data type** specifies the kind of data that the column stores. An Oracle10*g* database configures data columns using specific data types for two reasons. First, assigning a data type provides a means for error checking. For example, you cannot store the character data *Chicago* in a column assigned a DATE data type. Second, data types enable the DBMS to use storage space more efficiently by internally storing different types of data in different ways. The basic Oracle10*g* data types define character, number, date/time, and large object values.

Character Data Types

Character data columns store alphanumeric values that contain text and numbers not used in calculations, such as telephone numbers and postal codes. The Oracle10*g* character data types are VARCHAR2, CHAR, NVARCHAR2, and NCHAR. Each data type stores data in a different way.

NOTE

The Oracle10*g* database also supports the LONG data type, which allows you to store up to 2 gigabytes of character data. Because of limitations of the LONG data type, Oracle Corporation recommends that you store large volumes of character data using one of the large object data types rather than the LONG data type. Therefore, this book does not address the LONG data type.

VARCHAR2 Data Type

The **VARCHAR2** data type stores variable-length character data up to a maximum of 4000 characters. **Variable-length character data** is character data in which values in different rows can have a different number of characters. For example, in the Northwoods University database STUDENT table (see Figure 1-25), the values in the column that stores student last names may all have a different number of characters because people's names contain different numbers of characters.

You use the following syntax to declare a VARCHAR2 data column:

```
columnname VARCHAR2(maximum_size)
```

Columnname specifies the name of the column, and must follow the Oracle naming standard. *Maximum_size* can be an integer value from 1 to 4000. If the user inserts data values that are smaller than the specified maximum size, the DBMS stores only the actual character values. The DBMS does *not* add trailing blank spaces to the end of the entry to make the entry fill the maximum column size. If a user inserts a data value that has more characters than the maximum column size, an error occurs.

You would define the S_LAST column in the Northwoods University STUDENT table as follows:

```
s_last VARCHAR2(30)
```

This column declaration states that the column that stores a student's last name is a variable-length character column with a maximum of 30 characters. Examples of data stored in this column include the values Jones and Perez.

CHAR Data Type

The **CHAR** data type stores fixed-length character data up to a maximum of 2000 characters. **Fixed-length character data** is character data in which the data values for different rows all have the same number of characters. For example, the S_CLASS column in the Northwoods University STUDENT table stores values showing the student's current class standing (senior, junior, sophomore, or freshman). The S_CLASS column stores these values using the abbreviations SR, JR, SO, or FR, which all have exactly two characters.

The general syntax for declaring a CHAR data column is as follows:

```
columnname CHAR[(maximum_size)]
```

Maximum_size, which is optional, can be an integer value from 1 to 2000. If you omit the *maximum_size* value, the default size is one character. If a user inserts a data value in a CHAR column that has more characters than the maximum column size, an error occurs. If the user inserts data values that are smaller than the specified maximum size, the DBMS adds trailing blank spaces to the end of the entry to make the entry fill the *maximum_size* value. For example, if you declare the S_LAST column in the STUDENT table using the CHAR data type as s_last CHAR(30) and insert a data value of Perez, then the actual value that the database stores is Perez, plus 25 blank spaces to the right of the last character. Therefore, you should use the CHAR data type only when every data value in a column has exactly the same number of characters.

You would use the following command to define the S_CLASS column in the STUDENT table using the CHAR data type:

```
s_class CHAR(2)
```

The CHAR data type uses data storage space more efficiently than VARCHAR2, and the DBMS processes data retrieved from CHAR columns faster. However, if there is any chance that the data values might have different numbers of characters, use the VARCHAR2 data type.

NVARCHAR2 and NCHAR Data Types

Oracle10*g* stores character data in VARCHAR2 and CHAR columns using American Standard Code for Information Interchange (ASCII) coding, which represents each character as an 8-digit (one-byte) binary value. ASCII coding can represent a total of 256 different characters. As a result, you cannot use the VARCHAR2 or CHAR data types to represent data that is input in character sets other than standard English.

NOTE

Examples of other character sets are Kanji, which represents the Japanese language, and Cyrillic, which represents Russian and a variety of eastern European languages. To store these characters sets in an Oracle10*g* database, you must use Unicode coding.

To address the 256-character limitation of ASCII coding, the Oracle10*g* database provides the NVARCHAR2 and NCHAR data types. The **NVARCHAR2** and **NCHAR** data types store variable-length and fixed-length data just as their VARCHAR2 and CHAR counterparts do, except that they use Unicode coding. **Unicode** is a standardized technique that provides a way to encode data in diverse languages, and which is input using different character sets. The NVARCHAR2 data type is similar to the VARCHAR2 data type in all other ways, and the NCHAR data type is similar to the CHAR data type in all other ways. You declare NVARCHAR2 and NCHAR columns using the following general syntax:

```
columnname NVARCHAR2(maximum_size)
columnname NCHAR[(maximum_size)]
```

Number Data Types

Oracle10*g* stores all numerical data using the NUMBER data type. The NUMBER data type stores negative, positive, fixed, and floating-point numbers between 10^{-130} and 10^{125}, with precision up to 38 decimal places. **Precision** is the total number of digits both to the left and to the right of the decimal point.

NOTE

The precision value includes only the digits, and does not include formatting characters, such as the currency symbol, commas, or the decimal point.

You use the NUMBER data type for any column that stores numerical data upon which users may perform arithmetic calculations. (Recall that you use a character data type to store numerical data that will not be involved in calculations, such as telephone numbers or postal codes.) When you declare a NUMBER column, you optionally specify the precision and the scale using the following general syntax:

```
columnname NUMBER [([precision,] [scale])]
```

The *precision* value specifies the total number of digits in the data value, and the scale value specifies the number of digits on the right side of the decimal point. The precision value includes all digits, including the digits to the right of the decimal point, so the precision value is usually larger (it cannot be smaller) than the scale value. There are three NUMBER data subtypes: integer, fixed-point, and floating-point. You specify these data subtypes using different precision and scale configurations in the column declaration.

Integer Numbers

An integer is a whole number with no digits on the right side of the decimal point. To declare an integer data column, you omit the scale value in the data declaration using the following general syntax:

```
columnname NUMBER(precision)
```

You usually use integers to declare surrogate key columns. For example, the F_ID column in the FACULTY table stores integers (student identification numbers), so the declaration for this column is as follows:

```
f_id NUMBER(5)
```

In this column declaration, `NUMBER(5)` specifies that the F_ID column has a maximum length of five digits. The omission of the scale value specifies that the column values have no digits to the right of the decimal point. Examples of values in this column are 100, 101, 102, and so on.

For an integer data column, if the user enters a value that is smaller than the specified precision, the DBMS stores the actual data value, and does not store or display leading zeros. If the user attempts to enter a data value that is larger than the specified precision value, an error occurs. If the user enters a value that contains digits to the right of the decimal point, the DBMS rounds the value to the nearest whole number.

Fixed-point Numbers

A **fixed-point number** contains a specific number of decimal places, so the column declaration specifies both the precision and scale values. An example of a fixed-point data column is the PRICE column in the Clearwater Traders INVENTORY table. Data values in the PRICE column can range between 0 and 999.99, and always have exactly two digits to the right of the decimal point. You would use the following column declaration to define the PRICE column:

```
price NUMBER(5,2)
```

This declaration specifies that the PRICE column can have a maximum of five digits, with exactly two digits to the right of the decimal point. Examples of data values in the PRICE column are 259.99 and 59.99. Note that the decimal points are not included in the precision value that specifies the maximum width of the data value.

If the user attempts to enter a data value that is larger than the specified precision value, an error occurs. If the user enters a value that contains more digits to the right of the decimal point than specified in the scale, the DBMS rounds the value to the specified scale. For example, suppose the user inserts the value 19.579 in the PRICE column. Because the PRICE column has a scale of 2, the DBMS rounds the value to two decimal places, and stores the value as 19.58.

Floating-point Numbers

A **floating-point number** contains a variable number of decimal places. The decimal point can appear anywhere, from before the first digit to after the last digit, or can be omitted entirely. To define a floating-point data column, you omit both the precision and the scale values in the column declaration using the following syntax:

```
columnname NUMBER
```

Although no floating-point values exist in any of the case study databases, a potential floating-point column in the Northwoods database would be student grade point average (GPA), which you would declare as:

```
s_gpa NUMBER
```

A student's GPA might include one or more decimal places, such as 2.7045, 3.25, or 4.0.

Date and Time Data Types

The Oracle10*g* data types that store date and time values include the **datetime** data subtypes, which store actual date and time values, and the interval data subtypes, which store an elapsed time **interval** between two datetime values. The main datetime subtypes are DATE and TIMESTAMP. The interval subtypes include INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND. The following sections describe these data types.

DATE Data Type

The **DATE** data type stores dates from December 31, 4712 BC to December 31, AD 4712. The DATE data type stores the century, year, month, day, hour, minute, and second. The default date format is DD-MON-YY, which indicates the day of the month, a hyphen, the month (abbreviated using three capital letters), another hyphen, and the last two digits of the year. The default time format is HH:MI:SS AM, which indicates the hours, minutes, and seconds using a 12-hour clock. If the user does not specify a time when he or she enters a DATE data value, the default time value is 12:00:00 AM. If the user does not specify the date when he or she enters a time value, the default date value is the first day of the current month.

To declare a DATE data column, use the following general syntax:

```
columnname DATE
```

As always, *columnname* must adhere to the Oracle naming standard. Because the Oracle10*g* DBMS stores DATE columns in a standard internal format that does not vary, you do not need to include a length specification. An example of a data column that uses the DATE data type is the S_DOB column (student date of birth) in the Northwoods University STUDENT table, which you declare using the following command:

```
s_dob DATE
```

The S_DOB column stores a value such as 07-OCT-82 12:00:00 AM. The 12:00:00 AM time is the default time assigned to a date when you do not explicitly enter a time.

TIMESTAMP Data Type

The TIMESTAMP data type stores date values similar to the DATE data type, except it also stores fractional seconds in addition to the century, year, month, day, hour, minute, and second. An example of a TIMESTAMP data value is 15-AUG-06 09.26.01.123975 AM. You use the TIMESTAMP data type when you need to store precise time values. The following general syntax declares a TIMESTAMP data column:

```
columnname TIMESTAMP (fractional_seconds_precision)
```

In this syntax, *fractional_seconds_precision* specifies the precision (number of decimal places) that the DBMS will store for the fractional seconds. If you omit the *fractional_seconds_precision* specification, the default value is six decimal places.

For example, suppose you want to store a date value that includes the fractional seconds for when a shipment is received at Clearwater Traders, and you specify the fractional seconds using two decimal places. You would declare the SL_DATE_RECEIVED column as follows:

```
sl_date_received TIMESTAMP(2)
```

INTERVAL YEAR TO MONTH Data Type

Recall that the Oracle10*g* interval data types store an elapsed time interval between two dates. The INTERVAL YEAR TO MONTH data type stores a time interval, expressed in years and months, using the following syntax:

```
+|- elapsed_years-elapsed_months
```

In this syntax, +|- indicates that the interval can specify either a positive or negative time interval. If you add a positive time interval to a known date, the result is a date after the known date. If you add a negative time interval to a known date, the result is a date before the known date. *Elapsed_years* specifies the year portion of the interval, *elapsed_months* specifies the month portion of the interval, and the two values are separated by a hyphen (**-**). An example INTERVAL YEAR TO MONTH data value is +02-11, which specifies a positive time interval of 2 years and 11 months.

You use the following command to declare an INTERVAL YEAR TO MONTH data column:

```
columnname INTERVAL YEAR[(year_precision)] TO MONTH
```

In this syntax, *year_precision* specifies the maximum allowable number of digits that the column uses to express the year portion of the interval. (If you omit the *year_precision* value in the column declaration, the default value is 6, which enables the column to display a maximum interval of 999,999 years.) The TIME_ENROLLED column in the Northwoods University STUDENT table, which displays the amount of time that a student has been enrolled at the university, is an example of a column that uses an interval data type. You would use the following command to declare the TIME_ENROLLED column using the INTERVAL YEAR TO MONTH data type, with the default value of 6:

```
time_enrolled INTERVAL YEAR TO MONTH
```

INTERVAL DAY TO SECOND Data Type

The INTERVAL DAY TO SECOND data type stores a time interval, expressed in days, hours, minutes, and seconds. An INTERVAL DAY TO SECOND data column stores data values using the following general syntax:

```
+|- elapsed_days ↵
elapsed_hours:elapsed_minutes:elapsed_seconds
```

The following data value expresses a negative time interval of 4 days, 3 hours, 20 minutes, and 32 seconds: -04 03:20:32.00. The basic syntax to define an INTERVAL DAY TO SECOND data column is as follows:

```
columnname INTERVAL DAY[(leading_precision)] ↵
TO SECOND[(fractional_seconds_precision)]
```

In this syntax, *leading_precision* specifies the maximum allowable number of digits that the column uses to express the elapsed days, and *fractional_seconds_precision* specifies the maximum allowable number of digits that the column uses to express elapsed seconds. The default values for these expressions are 2 and 6, respectively.

You would use the following command to declare the TIME_ENROLLED column using the INTERVAL DAY TO SECOND data type and accept the default precision values:

```
time_enrolled INTERVAL DAY TO SECOND
```

Large Object (LOB) Data Types

Sometimes databases store binary data, such as digitized sounds or images, or references to binary files from a word processor or spreadsheet. In these cases, you can use one of the Oracle10*g* large object (LOB) data types. Table 2-1 summarizes the four LOB data types.

Large Object (LOB) Data Type	Description
BLOB	Binary LOB, storing up to 4 GB of binary data in the database
BFILE	Binary file, storing a reference to a binary file located outside the database in a file maintained by the operating system
CLOB	Character LOB, storing up to 4 GB of character data in the database
NCLOB	Character LOB that supports 2-byte character codes, stored in the database—up to a maximum of 4 GB

Table 2-1 Large object (LOB) data types

NOTE

Previous versions of Oracle supported the RAW and LONG RAW data types for storing binary data. Oracle Corporation recommends storing binary data using the large object data types, so this book does not use the RAW and LONG RAW data types.

You declare an LOB data column using the following general syntax:

```
columnname LOB_data_type
```

In this syntax, *LOB_data_type* is the name of the LOB data type, and can have the value BLOB, CLOB, BFILE, or NCLOB. Note that you do not specify the object size. The database automatically allocates the correct amount of space to store an LOB object when you insert the object into the database. For example, suppose you want to store the image of the faculty members in the F_IMAGE column of the Northwoods University FACULTY database table. You could store the actual binary image data using a BLOB data type, which you declare using the following command:

```
f_image BLOB
```

Alternately, you could store a reference to the location of an external image file using the BFILE data type by making the following declaration:

```
f_image BFILE
```

CONSTRAINTS

Constraints are rules that restrict the data values that you can enter into a column in a database table. There are two types of constraints: **integrity constraints**, which define primary and foreign keys; and **value constraints**, which define specific data values or data ranges that must be inserted into columns and whether values must be unique or not NULL. There are two levels of constraints: table constraints and column constraints.

A **table constraint** restricts the data value with respect to all other values in the table. An example of a table constraint is a primary key constraint, which specifies that a column value must be unique and cannot appear in this column in more than one table row.

A **column constraint** limits the value that can be placed in a specific column, irrespective of values that exist in other table rows. Examples of column constraints are value constraints, which specify that a certain value or set of values must be used, and NOT NULL constraints, which specify that a value cannot be NULL. A value constraint might specify that the value of a GENDER column must be either M (for male) or F (for female). You might place a NOT NULL constraint on a student ADDRESS column, to ensure that users always enter address information when they create a new customer row.

You can place constraint definitions at the end of the CREATE TABLE command, after you declare all of the table columns. Or, you can place each constraint definition within the column definition, so it immediately follows the data column declaration for the column associated with the constraint.

Every constraint in a user schema must have a unique constraint name that must adhere to the Oracle naming standard. You can explicitly define the constraint name, or omit the constraint name, and allow the Oracle10*g* DBMS to assign the constraint a system-generated name. The system-generated names are generic and use the prefix of SYS_C*n*, where *n* represents a numeric value designed to make each constraint name unique. This type of naming strategy makes it difficult to determine, solely by its name, with which table the constraint is associated. Therefore, it is a good practice to specify constraint names explicitly using descriptive names so you can easily identify the constraints that exist in your user schema.

A convention for assigning unique and descriptive constraint names is called the **constraint naming convention**. The constraint naming convention uses the following general syntax: `tablename_columnname_constraintID`. *Tablename* is the name of the table in which the constraint is defined, and *columnname* is the table column associated with the constraint. *ConstraintID* is a two-character abbreviation that describes the constraint. Table 2-2 shows commonly used *constraintID* abbreviations, and the Oracle10*g* constraint type identifier. (You will learn how to use the Oracle10*g* constraint type identifiers later in this chapter.)

Constraint Type	ConstraintID Abbreviation	Oracle10*g* Constraint Type Identifier
PRIMARY KEY	pk	P
FOREIGN KEY	fk	R
CHECK CONDITION	cc	C
NOT NULL	nn	N
UNIQUE	uk	U

Table 2-2 Common constraintID abbreviations

TIP

Sometimes the constraint naming convention causes constraint names to become longer than 30 characters. When this happens, you must abbreviate either the table name or the column name. Be sure to do this in a way that enables you to identify the table and column name with which the DBMS associates the constraint.

Integrity Constraints

An integrity constraint defines primary key columns, and specifies foreign keys and their corresponding table and column references. The following paragraphs describe how to define primary keys, foreign keys, and composite primary keys.

Primary Keys

The general syntax for defining a primary key constraint within a column declaration is:

```
CONSTRAINT constraint_name PRIMARY KEY
```

The syntax for defining a primary key constraint at the end of the CREATE TABLE command, after the column declarations, is:

```
CONSTRAINT constraint_name PRIMARY KEY (columnname)
```

Note that the first syntax, which defines the primary key within the column declaration, does not specify the column name. This is because the primary key constraint is defined within the column declaration. The second syntax includes the column name within the constraint definition, because the constraint is defined independently of the column declaration.

An example of a primary key column is LOC_ID in the Northwoods LOCATION table. You would use the following command to create the LOCATION table, and specify LOC_ID as the primary key (this example uses the syntax for which the constraint definition is independent of the column declarations, and uses the constraint naming convention to define the constraint name):

```
CREATE TABLE location
(loc_id NUMBER(6),
bldg_code VARCHAR2(10),
room VARCHAR2(6),
capacity NUMBER(5),
CONSTRAINT location_loc_id_pk PRIMARY KEY (loc_id));
```

You would use the following command to define LOC_ID as the table's primary key using the syntax for which the constraint definition is created within the column definition (note that in this approach, the column name is not repeated after the constraint definition):

```
CREATE TABLE location
(loc_id NUMBER(6)
CONSTRAINT location_loc_id_pk PRIMARY KEY,
bldg_code VARCHAR2(10),
room VARCHAR2(6),
capacity NUMBER(5));
```

Foreign Keys

A foreign key constraint is a column constraint that specifies that the value a user inserts in a column must exist as a primary key in a referenced table. As with primary keys, you can define foreign key constraints independently of a column declaration, or directly within the declaration of the column that has the constraint. The general syntax for specifying a foreign key constraint at the end of the column declarations and independent of a specific column declaration is:

```
CONSTRAINT constraint_name
FOREIGN KEY (columnname)
REFERENCES primary_key_tablename (primary_key_columnname)
```

Primary_key_tablename refers to the name of the table in which the foreign key is a primary key. *Primary_key_columnname* refers to the name of the primary key column in the *primary_key_tablename* table.

The general syntax for specifying a foreign key constraint within a column declaration is:

```
CONSTRAINT constraint_name
REFERENCES primary_key_tablename
(primary_key_columnname)
```

Note that the second syntax omits the keyword FOREIGN KEY and the constraint's column name. This is because the command declares the foreign key constraint within the column declaration, and the keyword REFERENCES implicitly defines the constraint as a foreign key constraint.

An example of a foreign key is the LOC_ID column in the Northwoods FACULTY table. The command to create the LOC_ID foreign key constraint in the FACULTY table independently of the column declaration is as follows:

```
CONSTRAINT faculty_loc_id_fk FOREIGN KEY (loc_id)
REFERENCES location (loc_id)
```

The command to create the LOC_ID foreign key constraint in the FACULTY table within the LOC_ID column declaration is as follows:

```
loc_id NUMBER(6) CONSTRAINT faculty_loc_id_fk
REFERENCES location (loc_id)
```

NOTE

Before you can create a foreign key reference, the table in which the column is a primary key must already exist, and the column must be defined as a primary key. In this example, you must create the LOCATION table and define LOC_ID as the primary key of the LOCATION table before creating the FACULTY table and defining LOC_ID as a foreign key within the FACULTY table.

Composite Keys

Recall that a composite key is a primary key composed of two or more data columns. You define a composite primary key constraint by listing all of the columns that make up the composite primary key, with each column name separated by a comma, using the following syntax:

```
CONSTRAINT constraint_name
PRIMARY KEY (columnname1, columnname2, …)
```

Constraint_name should consist of the table name, the names of all columns that make up the composite key, and the identifier pk. You must place the composite key definition after the commands in the CREATE TABLE command that define the columns that make up the key.

In the Northwoods ENROLLMENT table, the primary key consists of both the S_ID and the C_SEC_ID columns. The command to create the ENROLLMENT table and specify that the S_ID and C_SEC_ID columns make up the composite primary key is:

```
CREATE TABLE enrollment
(s_id NUMBER(5) CONSTRAINT enrollment_s_id_fk
REFERENCES student(s_id),
c_sec_id NUMBER(8) CONSTRAINT enrollment_c_sec_id_fk
REFERENCES course_section(c_sec_id),
CONSTRAINT enrollment_s_id_c_sec_id_pk
PRIMARY KEY (s_id, c_sec_id));
```

TIP

The columns that make up a composite key usually have foreign key constraints as well, as seen in the preceding example.

Value Constraints

Value constraints are column-level constraints that restrict the data values that users can enter into a given column. Commonly used value constraints include:

- CHECK conditions—This enables you to specify that a column value must be a specific value or fall within a range of values.
- NOT NULL constraint—Specifies whether a column value can be NULL
- DEFAULT constraint—Specifies that a column has a default value that the DBMS automatically inserts for every row, unless the user specifies an alternate value
- UNIQUE constraint—Specifies that a column must have a unique value for every table row

Check Conditions

Check conditions specify that a column value must be a specific value (such as M), from a set of allowable values (such as M or F), or fall within a specific range (such as greater than zero but less than 1000). You should create check condition constraints only when the number of allowable values is limited and not likely to change. You must be prudent when specifying check conditions in table definitions, because once the table is populated with data, it is difficult or impossible to modify the constraint—all rows must satisfy the constraint.

The DBMS must be able to evaluate each expression in a check condition as either true or false. You can combine expressions using the logical operators AND and OR. When you join two expressions using the AND operator, both expressions must be true for the expression to be true. When you join two expressions using the OR operator, only one expression needs to be true for the expression to be true.

As an example of a check condition, consider the S_CLASS column in the Northwoods University STUDENT table, in which the values are restricted to FR, SO, JR, or SR (freshman, sophomore, junior, or senior). The syntax to define this check condition is:

```
CONSTRAINT student_s_class_cc CHECK
((s_class = 'FR') OR (s_class = 'SO')
OR (s_class = 'JR') OR (s_class ='SR'))
```

You can also use a check condition to validate a range of allowable values. An example of a range check condition is in the CREDITS column in the Northwoods COURSE table, where the allowable values must be greater than 0 and less than 12. The constraint definition is:

```
CONSTRAINT course_credits_cc
CHECK((credits > 0) AND (credits < 12))
```

Note that the DBMS can evaluate both of the expressions `(credits > 0)` and `(credits < 12)` as either true or false. The AND condition specifies that both conditions must be true to satisfy the check condition.

NOT NULL Constraints

The **NOT NULL constraint** specifies whether the user must enter a value for a specific row, or whether the value can be NULL (absent or unknown). Primary key columns automatically have a NOT NULL constraint. From a business standpoint, some columns, such as a customer's name, should not be NULL. The following syntax specifies that the S_LAST column in the Northwoods student table must contain a value or the row is rejected, rather than added to the table:

```
s_last VARCHAR2(30)
CONSTRAINT student_s_last_nn NOT NULL
```

Default Constraints

A **default constraint** specifies that a particular column has a default value that the DBMS automatically inserts for every table row. This constraint must be created in the column declaration, not as a separate command beginning with CONSTRAINT. For example, if most Northwoods University students live in Florida, you would use the following column declaration to specify that the S_STATE column has a default value of FL. If the default value is a character string, you must enclose the value in single quotation marks.

```
s_state CHAR(2) DEFAULT 'FL'
```

The DBMS will insert the default only if the user inserts a NULL value into the S_STATE column. If the user specifies an alternate value, the alternate value replaces the default value.

UNIQUE Constraints

A **UNIQUE constraint** is a table constraint that specifies that a column must have a unique value for every table row. It is basically the same as a primary key constraint, except NULL values are allowed in the column. NULL values are not allowed in a column that is referenced by a primary key constraint. An example of a column that might require a unique constraint is the TERM_DESC column in the Northwoods University TERM table, to specify that each term has a unique description. You would use the following command to create this constraint:

```
CONSTRAINT term_term_desc_uk UNIQUE (term_desc)
```

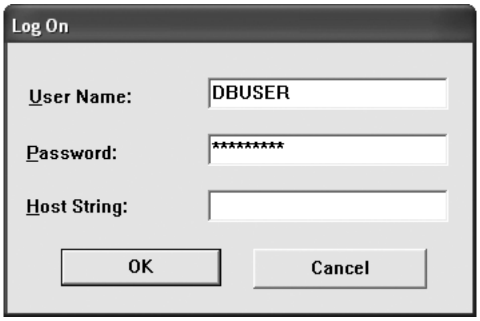
CREATING DATABASE TABLES USING SQL*PLUS

This next section explains how to create database tables using SQL*Plus, Oracle10*g*'s command-line SQL utility. Starting an Oracle10*g* client application such as SQL*Plus is a two-step process: first you start the application on your client workstation, then you log onto the Oracle10*g* database. In this first set of steps, you start SQL*Plus, log onto the database, and create some database tables.

NOTE If you are working in a computer laboratory, your instructor will inform you of the correct procedures to start and log onto the Oracle10*g* database. If you are using Oracle10*g* Personal Edition, use the logon procedures in the installation instructions.

To start SQL*Plus:

1. Click **Start** on the Windows taskbar, point to **All Programs** (if you are using Windows 2000, point to **Programs**), point to **Oracle10*g* OraHome10**, point to **Application Development**, and then click **SQL Plus**. The Log On dialog box opens and requests your username, password, and host string, as shown in Figure 2-1.


Log On

User Name: DBUSER

Password: *********

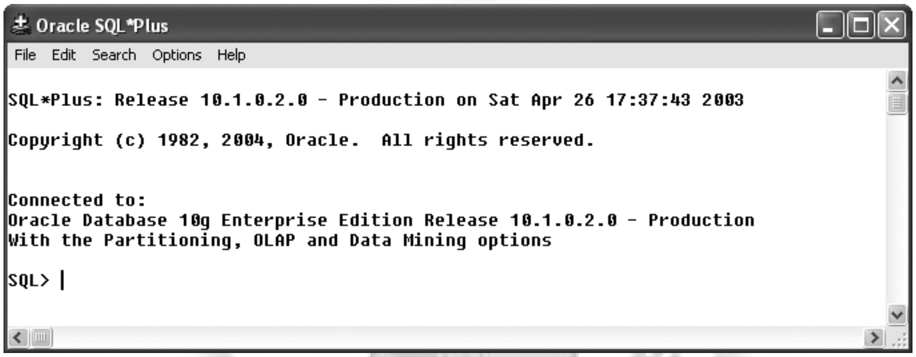
Host String:

Figure 2-1 SQL*Plus Log On dialog box

NOTE

These instructions assume that you installed the Oracle10*g* DBMS software in the OraHome10 folder. If you installed the software in a different folder, point to **All Programs**, then point to **Oracle10*g* *foldername*** instead. If you installed the program on your home machine and have created only one database, you can leave the Host String blank. However, if there is more than one database, or if you receive a TNS error, enter the name of your database as the Host String.

2. Type your username, press **Tab**, type your password, press **Tab** again, type the host string provided by your instructor, and then click **OK**. (If you are using Oracle10*g* Personal Edition, leave the host string blank.) The SQL*Plus program window opens, as shown in Figure 2-2. If necessary, maximize the program window.



```
SQL*Plus: Release 10.1.0.2.0 - Production on Sat Apr 26 17:37:43 2003

Copyright (c) 1982, 2004, Oracle.  All rights reserved.


Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> |
```

Figure 2-2 SQL*Plus window

HELP

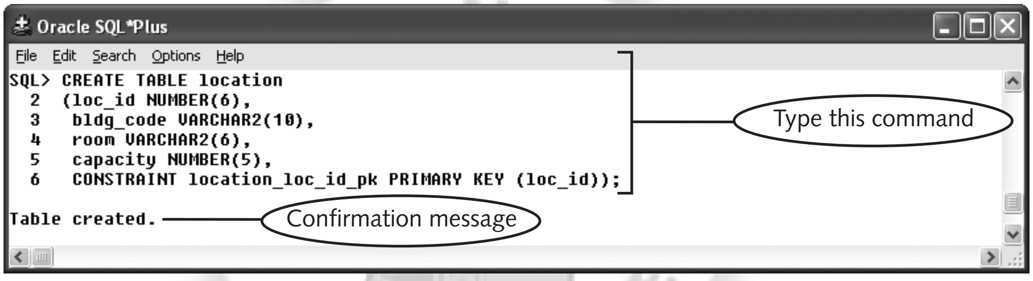
Your release of SQL*Plus might have a different version number, depending on the client or server software you are using. The steps in the book will work the same regardless of the version number.

To use SQL*Plus, you type SQL commands at the SQL prompt. You end each command with a semicolon (;), which marks the end of the command. Then you press Enter to submit the command to the SQL*Plus interpreter. The **SQL*Plus interpreter** checks the command for syntax errors, and if the command is error free, the SQL interpreter submits the command to the database. SQL commands are not case sensitive, and the SQL interpreter ignores not only spaces between characters but line breaks as well. However, it is a good practice to format your SQL commands using uppercase letters for reserved words, and to break the command across multiple lines, so the command is easier to understand and debug.

When you create database tables that contain foreign key references to other tables, you must first create the table in which the foreign key is a primary key. For example, to create the Northwoods University STUDENT table, you must first create the FACULTY table, because the F_ID column in the STUDENT table is a foreign key that references the F_ID column in the FACULTY table. However, note that the FACULTY table contains the LOC_ID column, which is a foreign key that references the LOC_ID column in the LOCATION table. The LOCATION table has no foreign key references, so you can create the LOCATION table before you create any other tables. (In other words, you must create the LOCATION table before you can create the FACULTY table, and you must create the FACULTY table before you can create the STUDENT table.) Now you will create the LOCATION table.

To create the LOCATION table:

1. In SQL*Plus, type the command in Figure 2-3 at the SQL prompt. Do not type the line numbers. SQL*Plus adds line numbers to your command after you press Enter. Each line of the command defines a different column of the database.



```
SQL> CREATE TABLE location
  2  (loc_id NUMBER(6),
  3   bldg_code VARCHAR2(10),
  4   room VARCHAR2(6),
  5   capacity NUMBER(5),
  6   CONSTRAINT location_loc_id_pk PRIMARY KEY (loc_id));

Table created.
```

Figure 2-3 SQL command to create the LOCATION table

2. Type **;** as the last character of the last line to end the command. The semicolon marks the end of the SQL command for the SQL interpreter.

3. Press **Enter** to execute the command. The confirmation message "Table created" appears, as shown in Figure 2-3.

HELP

If your command had an error, the next section will describe how you can edit the command using a text editor.

Creating and Editing SQL Commands Using a Text Editor

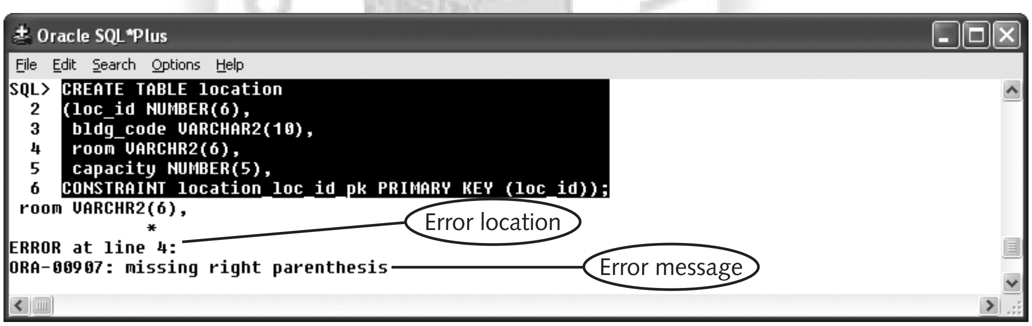
Many SQL commands are long and complex, and it is easy to make typing errors. A good approach for entering and editing SQL*Plus commands is to type commands into a text editor such as Notepad, then copy your commands, paste the copied commands into SQL*Plus, and execute the commands. If the command has an error, you can switch back to the text editor, edit the command, copy and paste the edited text back into SQL*Plus, and then reexecute the command.

When you are creating database tables, it is a good idea to save the text of all of your CREATE TABLE commands in a single Notepad text file so you have a record of the original code. Saving all the commands in one file creates a **script**, which is a text file that contains several related SQL commands. You can run the script later to re-create the tables if you need to make changes. You can save multiple CREATE TABLE commands in such a text file. Just make sure that they are in the proper order so that foreign key references are made after their parent tables are created.

Next you learn how to use Notepad to edit SQL*Plus commands. If your SQL*Plus commands have errors, you can edit the commands, and reexecute them. You also want to save the Notepad file so you have a record of your commands. This is the beginning of the script file that creates the tables in the Northwoods University database. You continue to use this file to record and edit your SQL commands for the rest of this chapter.

To edit the command in Notepad and save the file:

1. In SQL*Plus, type the command in Figure 2-4 at the SQL prompt, exactly as shown. (This command purposely contains an error.) Press **Enter** to execute the command. An error message appears as shown in Figure 2-4 because the command that declares the ROOM column spells VARCHAR2 as VARCHR2.
2. Select the text as shown in Figure 2-4. Do not select the line numbers, the SQL prompt, confirmation, or error message.



```
SQL> CREATE TABLE location
  2  (loc_id NUMBER(6),
  3   bldg_code VARCHAR2(10),
  4   room VARCHR2(6),
  5   capacity NUMBER(5),
  6  CONSTRAINT location_loc_id_pk PRIMARY KEY (loc_id));
 room VARCHR2(6),
           *
ERROR at line 4:
ORA-00907: missing right parenthesis
```

Figure 2-4 Selecting the SQL command text

3. Click **Edit** on the menu bar, and then click **Copy**.

TIP

Another way to copy text is to highlight the text, and then press Ctrl+C. (The C is not case sensitive.)

4. Start Notepad (or an alternate text editor) on your computer.
5. In Notepad, click **Edit** on the menu bar, and then click **Paste**. The SQL command text appears in Notepad.

TIP

Another way to paste text is to press Ctrl+V. (The *V* is not case sensitive.)

6. Correct the command text by changing the data type spelling to VARCHAR2 in the ROOM column declaration. Then copy the command, switch to SQL*Plus, paste the corrected text at the SQL prompt, and press **Enter** to run the command again. (If you successfully created the LOCATION table the first time, an error message stating "name is already used by an existing object" appears.) If a different error message appears, and you still have not yet successfully created the LOCATION table, switch back to Notepad. Debug the command, and run the command again until the message appears confirming that the LOCATION table has been successfully created.
7. Switch back to Notepad, click **File**, and then click **Save**. Navigate to the Chapter2\Tutorials folder on your Data Disk, type **Ch2Queries.sql** in the File name text box, open the Save as type list, select **All files**, and then click **Save**.

TIP

Script files normally have a .sql extension. If you do not open the Save as type list and select All Files, Notepad automatically appends a .txt extension onto the filename.

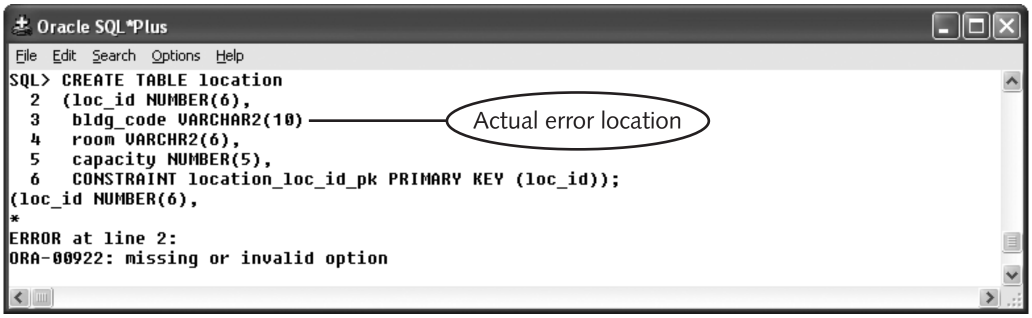
8. Minimize Notepad.

Using Oracle Online Help Resources to Debug SQL Commands

When a SQL command contains a syntax error, the SQL*Plus interpreter displays error information that includes the line number within the command that caused the error, the position of the error within the line, and an error code and description of the error. When you tried to create the LOCATION table, the error message shown in Figure 2-4 appeared. Note that the interpreter displays the command that contains the error, and shows the position of the error in the command by placing an asterisk (*) under the character that caused the error.

Oracle10*g* error codes have a 3-character prefix (such as ORA), and a 5-digit error code. The prefix indicates which Oracle10*g* utility generated the error. In this case, the ORA prefix indicates that the error was generated by the DBMS. The error code (00907) is a numeric code assigned by Oracle Corporation that indicates that a right (closing) parenthesis was omitted in a command. In this case, the data type for the ROOM column was misspelled as VARCHR2 instead of VARCHAR2. Although the error description ("missing right parenthesis") does not accurately describe the error, the asterisk flagging the error position makes the error easy to locate.

Sometimes the causes of SQL command errors are not readily apparent, and you need to retrieve more information about the error. For example, consider the error message in Figure 2-5.



```
SQL> CREATE TABLE location
  2  (loc_id NUMBER(6),
  3   bldg_code VARCHAR2(10)
  4   room VARCHR2(6),
  5   capacity NUMBER(5),
  6   CONSTRAINT location_loc_id_pk PRIMARY KEY (loc_id));
(loc_id NUMBER(6),
*
ERROR at line 2:
ORA-00922: missing or invalid option
```

Figure 2-5 SQL command error message requiring further explanation

The error message indicates that the error occurred on Line 2, and the asterisk indicates that the error occurred at the position of the opening parenthesis before the first table column (LOC_ID) was specified. When you inspect the second line of the command, the line does not appear to have any errors. The error code explanation ("missing or invalid option") does not adequately explain the error cause, nor does it provide ideas for locating or correcting the error.

To get further information about this error, you can connect to the Oracle Technology Network (OTN) Web site and search for the error code. OTN is a Web-based resource that Oracle Corporation provides free of charge.

Although the format of the Web site tends to change, usually you can view reference material regarding Oracle error codes by selecting the Documentation option available on the home page. When the Documentation page appears, select the option to view the Documentation available for the Oracle10*g* Database. After accessing the database documentation portion of the Web site, you can perform a search on the specific error code returned by the DBMS (error code ORA-00922 in this example). Documentation for most ORA- error codes returns the possible causes for the error and action necessary to correct the problem.

If a search for the error code does not return the desired results, you can also perform a search for "Part Number B10744-01" which provides access to Oracle® Database Error Messages 10*g* Release 1 (10.1) available on the Web site. Each chapter provides access to a different range of error codes. A list of the different chapters is provided on the Contents page.

When an error occurs that you cannot locate, a last resort debugging technique is to create the table multiple times, each time adding a column declaration repeating the process until you find the declaration causing the error. First, paste your nonworking command in a Notepad file and modify it so that it creates the table with only the first column declaration. Copy the modified command, and paste it into SQL*Plus. If SQL*Plus successfully creates the table with the first column, you now know that the error is not in the first column declaration. Delete the table using the DROP TABLE command, which has the following syntax: `DROP TABLE` *`tablename`*`;`. (You will learn more about the DROP TABLE command later in this chapter.) Then, modify the command in Notepad to create the table using only the first and second column declarations. If this works, you now know that the problem is not in either the first or second column declaration. Drop the table again, and modify the command to create the table using only the first, second, and third column declarations. Continue this process of adding one more column declaration to the CREATE command until you locate the column declaration that is causing the error.

NOTE

If you find the information regarding ORA error messages posted on the OTN confusing or obscure, you can also use a search engine such as Google to locate information posted by other users at different locations on the Internet.

Exiting SQL*Plus

There are three ways to exit SQL*Plus:

- Type exit at the SQL prompt.
- Click File on the menu bar, and then click Exit.
- Click the Close button on the program window title bar.

You should *never* exit SQL*Plus by simply shutting down the machine. This can lead to corrupt files and prevent you from starting the database in the future.

NOTE

When entering or editing data that is stored in a database table, you should not click the Close button to exit SQL*Plus or you risk losing any uncommitted data changes.

Your database connection disconnects automatically when you exit SQL*Plus. In the next set of steps, you exit SQL*Plus by typing exit at the SQL prompt.

To exit SQL*Plus:

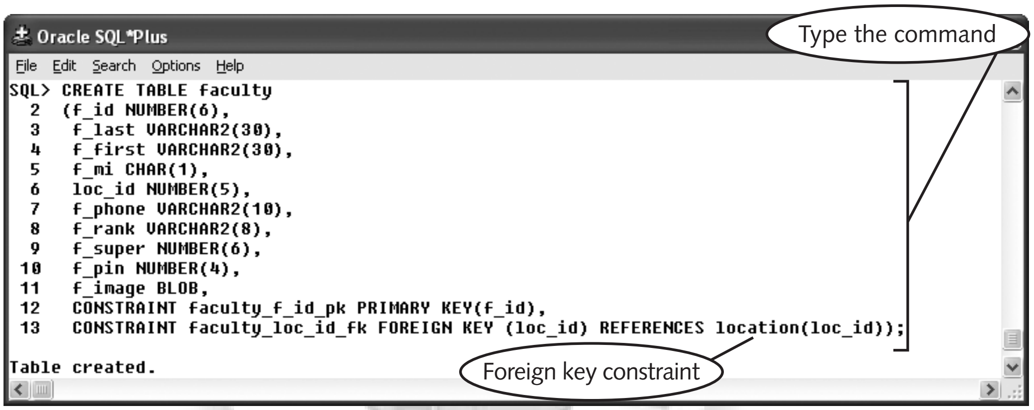
1. Type **exit** at the SQL prompt.
2. Press **Enter**. Your database connection disconnects, and the SQL*Plus window closes.

Creating a Table with a Foreign Key Constraint

Recall that a foreign key creates a relationship between two database tables. In the Northwoods University FACULTY table, LOC_ID is a foreign key that references a row in the LOCATION table. Now you create the FACULTY table, and specify the LOC_ID column as a foreign key.

To create the FACULTY table and specify the foreign key:

1. Switch to Notepad, where your Ch2Queries.sql file should be open. If necessary, press **Enter** to create a new blank line, and then type the command in Figure 2-6 to create the FACULTY table.



```
SQL> CREATE TABLE faculty
  2  (f_id NUMBER(6),
  3   f_last VARCHAR2(30),
  4   f_first VARCHAR2(30),
  5   f_mi CHAR(1),
  6   loc_id NUMBER(5),
  7   f_phone VARCHAR2(10),
  8   f_rank VARCHAR2(8),
  9   f_super NUMBER(6),
 10   f_pin NUMBER(4),
 11   f_image BLOB,
 12   CONSTRAINT faculty_f_id_pk PRIMARY KEY(f_id),
 13   CONSTRAINT faculty_loc_id_fk FOREIGN KEY (loc_id) REFERENCES location(loc_id));

Table created.
```

Figure 2-6 SQL command to create the FACULTY table

2. Select and copy all of the CREATE TABLE faculty command text.
3. Start SQL*Plus, log on to the database, paste the copied command at the SQL prompt, and then press **Enter**. If necessary, debug the command until you successfully create the FACULTY table.

NOTE For the rest of the chapter, you should type SQL commands in the 2Queries.sql file, unless the instructions state to type the command at the SQL prompt. Then paste the commands into SQL*Plus, and debug the commands if necessary.

VIEWING INFORMATION ABOUT TABLES

After you create database tables, you often need to review information such as the table name, column and data type values, and constraint types. For example, when you write a command to insert a new row, you need to specify the table name, and you need to list the data values in the same order as the table column names. To view the column names and data types of an individual table, you use the DESCRIBE command, which has the following syntax:

```
DESCRIBE tablename
```

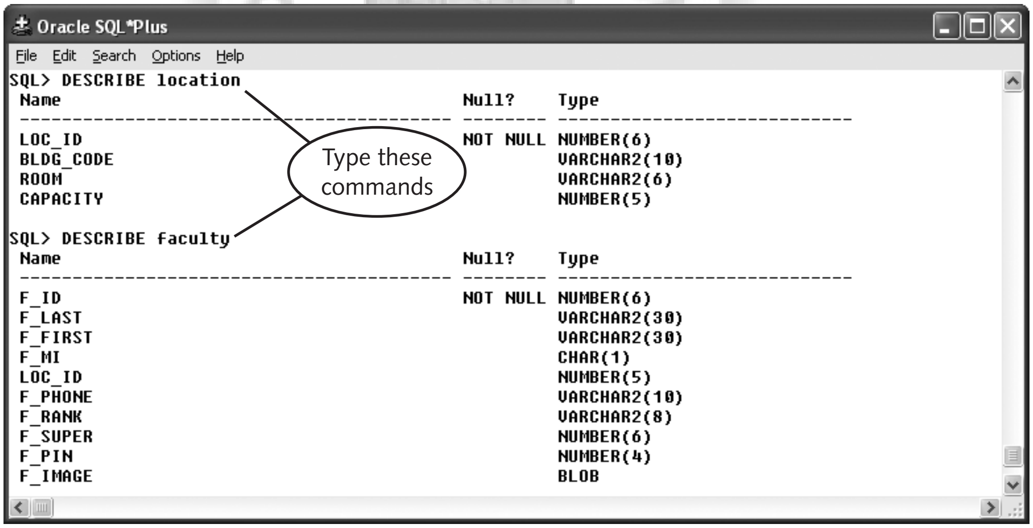
Next you use the DESCRIBE command to view information about the columns in the LOCATION and FACULTY tables that you created earlier.

NOTE

The DESCRIBE command is actually a SQL*Plus command and not a SQL command so it does not require a semicolon at the end of the command to execute. Another feature of SQL*Plus commands is that they can be abbreviated. For example, you can type DESC *tablename* rather than DESCRIBE *tablename* to view the structure of a table.

To view the table columns:

1. In SQL*Plus, type the command **DESCRIBE location**, and then press **Enter**. The column names and data types for the LOCATION table should appear as shown in Figure 2-7.
2. Next, type **DESCRIBE faculty** at the SQL prompt and then press **Enter**. The column names and data types for the FACULTY table appear as shown in Figure 2-7.



```
SQL> DESCRIBE location
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 LOC_ID                                    NOT NULL NUMBER(6)
 BLDG_CODE                                          VARCHAR2(10)
 ROOM                                               VARCHAR2(6)
 CAPACITY                                           NUMBER(5)

SQL> DESCRIBE faculty
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 F_ID                                      NOT NULL NUMBER(6)
 F_LAST                                             VARCHAR2(30)
 F_FIRST                                            VARCHAR2(30)
 F_MI                                               CHAR(1)
 LOC_ID                                             NUMBER(5)
 F_PHONE                                            VARCHAR2(10)
 F_RANK                                             VARCHAR2(8)
 F_SUPER                                            NUMBER(6)
 F_PIN                                              NUMBER(4)
 F_IMAGE                                            BLOB
```

Figure 2-7 Using the DESCRIBE command to view table columns

Note that the output lists the column names and data types. It also shows the columns that have NOT NULL constraints: LOC_ID in the LOCATION table and F_ID in the FACULTY table. These columns cannot be NULL because they are the tables' primary key columns. However, note that the DESCRIBE command does not list the foreign key constraint on the LOC_ID column in the FACULTY table. To view information about constraints other than the NOT NULL constraint, you use the Oracle10*g* data dictionary views.

The **Oracle10*g* data dictionary** consists of tables that contain information about the structure of the database. When the DBA creates a new database, the system creates the data dictionary in a user schema named SYS. The Oracle10*g* DBMS automatically updates the data dictionary tables as users create, update, and delete database objects. As a general rule, users and database administrators do not directly view, update, or delete values in the data dictionary tables. Rather, users interact with the data dictionary using the **data dictionary views**. A **view** is a database object that the DBMS bases on an actual database table and which enables the DBMS to present the table data in a different format based on the needs of users. A view can serve to hide some table columns in which the user has no interest or doesn't have required privileges to view.

The data dictionary views are divided into three general categories:

- USER—Shows the objects in the current user's schema
- ALL—Shows both the objects in the current user's schema and the objects that the user has privileges to manipulate. For example, another user might create a table, and then give you the privilege to update the table.
- DBA—Allows users who are database administrators to view information about all database objects

The general command to retrieve information from a data dictionary view is:

```
SELECT view_columnname1, view_columnname2, …
FROM prefix_object;
```

NOTE

You learn more about using the SELECT command to retrieve database data in Chapter 3. In this chapter, you will use the SELECT command in a limited way to retrieve information about your database tables.

In this syntax, *view_columnname1*, *view_columnname2,* and so forth reference the names of the columns in the view that you wish to retrieve. *Prefix* specifies the view category, and can have the value USER, DBA, or ALL. *Object* is the type of database object you are examining, such as TABLES or CONSTRAINTS. For example, the following command retrieves the names of all of a user's database tables by retrieving the TABLE_NAME column from the USER_TABLES view:

```
SELECT table_name
FROM user_tables;
```

Similarly, the following command uses the ALL prefix and retrieves the names of all database tables that a user has either created or has been given object privileges to manipulate:

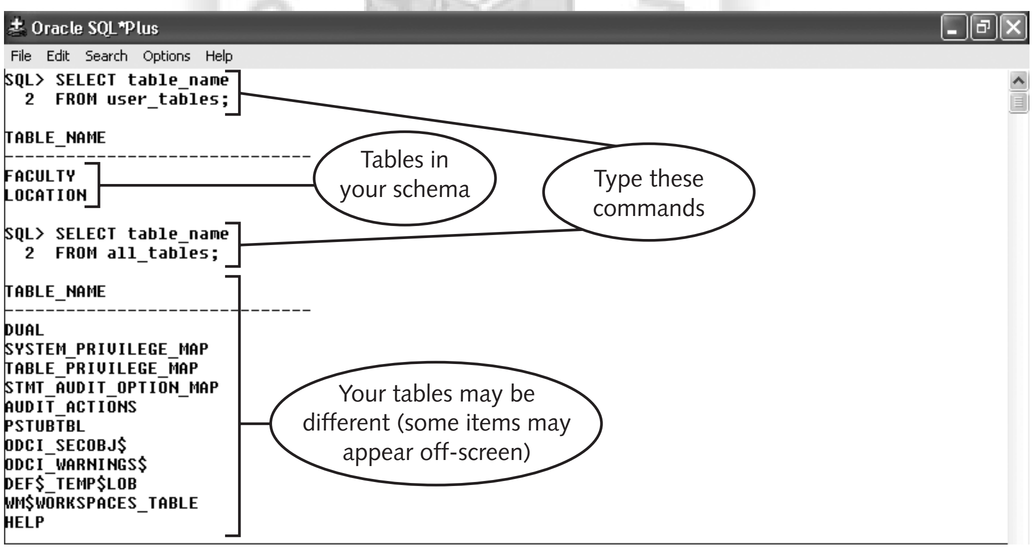
```
SELECT table_name
FROM all_tables;
```

In the next set of steps, you use data dictionary view commands to view information about your tables.

To view information about your tables:

1. Type **`SELECT table_name FROM user_tables;`** at the SQL prompt, and then press **Enter** to view information about the tables in your user schema. The output should be similar to the output in Figure 2-8. Your output will be different if you have created additional tables.

2. Type **`SELECT table_name FROM all_tables;`** at the SQL prompt, and then press **Enter** to retrieve the names of all tables that you have privileges to manipulate. Again, the output should appear similar to the output shown in Figure 2-8. (The output of this command will be different if other database users have created tables and given you privileges to manipulate these tables. Some of the output may scroll off the screen. The LOCATION and FACULTY tables are at the bottom of the list and are not shown in Figure 2-8.)


Oracle SQL*Plus
File Edit Search Options Help
Tables in your schema
Type these commands
Your tables may be different (some items may appear off-screen)

```
SQL> SELECT table_name
  2  FROM user_tables;

TABLE_NAME
------------------------------
FACULTY
LOCATION

SQL> SELECT table_name
  2  FROM all_tables;

TABLE_NAME
------------------------------
DUAL
SYSTEM_PRIVILEGE_MAP
TABLE_PRIVILEGE_MAP
STMT_AUDIT_OPTION_MAP
AUDIT_ACTIONS
PSTUBTBL
ODCI_SECOBJ$
ODCI_WARNINGS$
DEF$_TEMP$LOB
WM$WORKSPACES_TABLE
HELP
```

Figure 2-8 Retrieving table names from the USER_TABLES data dictionary view

You can retrieve information about a variety of database objects using different data dictionary views. Table 2-3 shows the names of database objects for which you can retrieve information from data dictionary views.

Object Name	Object Type
OBJECTS	All database objects
TABLES	Database tables
INDEXES	Table indexes created to improve query retrieval performance
VIEWS	Database views
SEQUENCES	Sequences used to generate surrogate key values automatically
USERS	Database users
CONSTRAINTS	Table constraints
CONS_COLUMNS	Table columns that have constraints
IND_COLUMNS	Table columns that have indexes
TAB_COLUMNS	All table columns

Table 2-3 Database objects with data dictionary views

You can retrieve information from these views using commands similar to the ones you used to retrieve information about your database tables. For example, the command to retrieve the names of all of your database objects is:

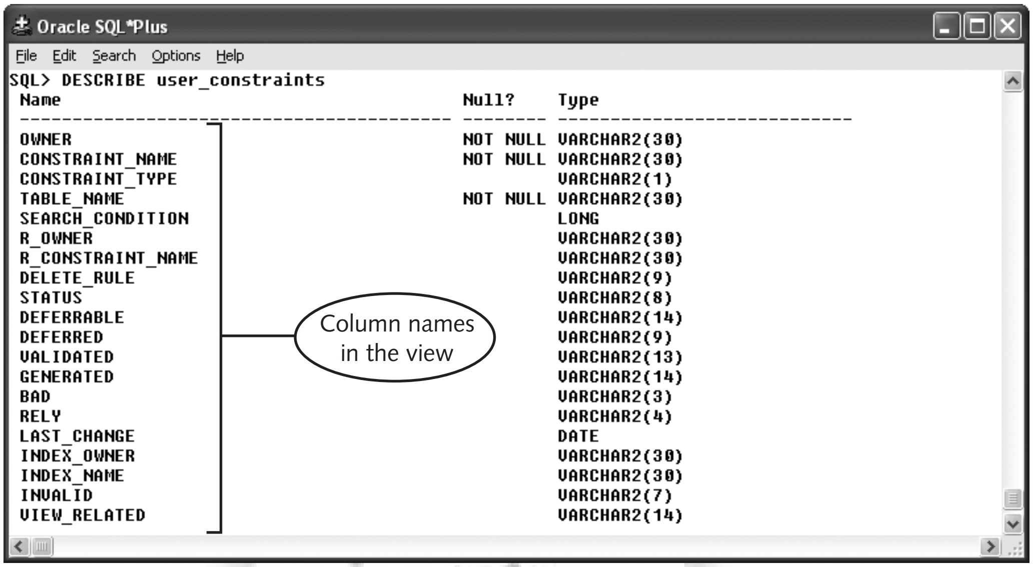
```
SELECT object_name
FROM user_objects;
```

All the data dictionary views have different columns. To determine the names of the columns in a specific database view, you use the DESCRIBE command along with the view name. Figure 2-9 shows a description of the column names in the USER_CONSTRAINTS data dictionary view.

Next you query the USER_CONSTRAINTS view to determine the constraints that exist in your database tables. You will select the CONSTRAINT_NAME, TABLE_NAME, and CONSTRAINT_TYPE columns from the USER_CONSTRAINTS view.

To view your table constraints:

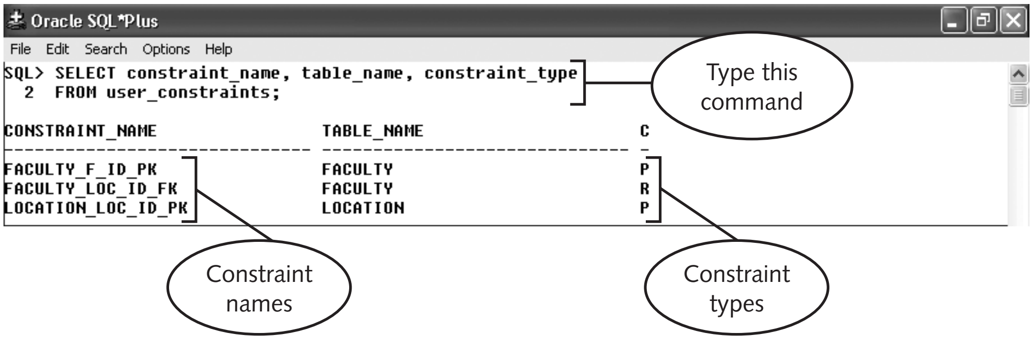
1. In SQL*Plus, type the command in Figure 2-10 at the SQL prompt as shown.
2. Press **Enter** to execute the command. Figure 2-10 shows the output. Your output may be different if you have created additional database tables with other constraints.



```
SQL> DESCRIBE user_constraints
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 OWNER                                     NOT NULL VARCHAR2(30)
 CONSTRAINT_NAME                           NOT NULL VARCHAR2(30)
 CONSTRAINT_TYPE                                    VARCHAR2(1)
 TABLE_NAME                                NOT NULL VARCHAR2(30)
 SEARCH_CONDITION                                   LONG
 R_OWNER                                            VARCHAR2(30)
 R_CONSTRAINT_NAME                                  VARCHAR2(30)
 DELETE_RULE                                        VARCHAR2(9)
 STATUS                                             VARCHAR2(8)
 DEFERRABLE                                         VARCHAR2(14)
 DEFERRED                                           VARCHAR2(9)
 VALIDATED                                          VARCHAR2(13)
 GENERATED                                          VARCHAR2(14)
 BAD                                                VARCHAR2(3)
 RELY                                               VARCHAR2(4)
 LAST_CHANGE                                        DATE
 INDEX_OWNER                                        VARCHAR2(30)
 INDEX_NAME                                         VARCHAR2(30)
 INVALID                                            VARCHAR2(7)
 VIEW_RELATED                                       VARCHAR2(14)
```

Figure 2-9 Column names in the USER_CONSTRAINTS data dictionary view

The output shows that your user schema has three constraints, two in the FACULTY table and one in the LOCATION table. The constraint names are the values you specified using the constraint naming convention when you created the tables. These values make it easy to determine a constraint's associated column name, and the constraint identifiers (`pk` and `fk`) identify the type of each constraint (primary key and foreign key). The final column in the query output also shows the constraint types. Recall that Table 2-2 shows the letter that Oracle10*g* uses to identify different constraint types. In Figure 2-10, the CONSTRAINT_TYPE column identifies the primary key constraints using the letter *P*, and the foreign key constraints using the letter *R*. In Figure 2-10, the CONSTRAINT_TYPE column identifies the primary key constraints using the letter P and the foreign key constraints using the letter R, which stands for REFERENCE.



```
SQL> SELECT constraint_name, table_name, constraint_type
  2  FROM user_constraints;

CONSTRAINT_NAME                TABLE_NAME                     C
------------------------------ ------------------------------ -
FACULTY_F_ID_PK                FACULTY                        P
FACULTY_LOC_ID_FK              FACULTY                        R
LOCATION_LOC_ID_PK             LOCATION                       P
```

Figure 2-10 Viewing table constraint information

NOTE The "C" displayed as the heading for the CONSTRAINT_TYPE column in Figure 2-10 is limited to the single letter *C* because the data in the column is declared as a VARCHAR2 of size 1. If the number of characters declared for a column is less than the number of characters in the name of the column, the column name in the heading is truncated to the number of characters in the column declaration.

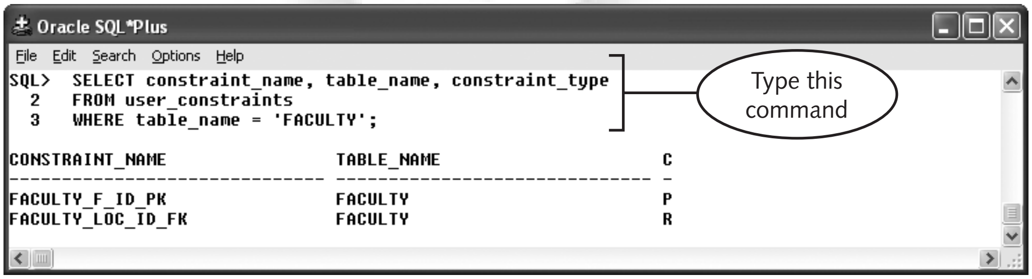
To retrieve a list of all of the constraints for a specific database table in your user schema, you use the following command:

```
SELECT constraint_name, constraint_type
FROM user_constraints
WHERE table_name = 'DATABASE_TABLENAME';
```

In this syntax, the *DATABASE_TABLENAME* parameter is the name of the table for which you want to display constraints. You must specify this value in all uppercase letters because the data dictionary stores all object information in uppercase letters. You also must enclose the value in single quotation marks because it is a character string. Now you query the USER_CONSTRAINTS data dictionary view to retrieve information about the constraints in the FACULTY table.

To list the FACULTY table constraints:

1. In SQL*Plus, type the command shown in Figure 2-11 to retrieve information about the constraints in the FACULTY table.
2. Press **Enter** to execute the command. Your output should look similar to the output in Figure 2-11. (Don't worry if your constraints appear in a different order than the ones listed in Figure 2-11. You learn how to change the order of retrieved data values in Chapter 3.)



```
SQL>  SELECT constraint_name, table_name, constraint_type
  2   FROM user_constraints
  3   WHERE table_name = 'FACULTY';

CONSTRAINT_NAME                TABLE_NAME                     C
------------------------------ ------------------------------ -
FACULTY_F_ID_PK                FACULTY                        P
FACULTY_LOC_ID_FK              FACULTY                        R
```

Figure 2-11 Viewing information about constraints in a specific table

MODIFYING AND DELETING DATABASE TABLES

When you design a database, you should plan your tables carefully to avoid having to change the structure of the database tables later. When you change database tables, often the applications that retrieve information from the tables no longer work correctly and have to be modified. However, Oracle10*g* provides techniques for modifying existing database tables if you need to do so.

In Oracle10*g*, you can modify existing database tables by performing actions such as changing the name of a table, adding new columns, deleting columns that are no longer needed, or changing the data type or maximum size of an existing column. There are some specifications of an Oracle10*g* database table that you can always modify. To modify these specifications, you perform an **unrestricted action**. There are other table specifications that you can modify only in certain situations. To modify these specifications, you perform a **restricted action**. Table 2-4 lists unrestricted actions, and Table 2-5 summarizes restricted actions when modifying Oracle10*g* database tables.

Unrestricted Actions
Renaming a table
Adding new fields
Deleting fields
Increasing the *maximum_size* value of a field
Deleting constraints

Table 2-4 Unrestricted actions when modifying database tables

The following sections describe how to delete and rename existing tables, add columns to existing tables, modify existing column data definitions, delete columns, and modify existing column constraints.

Deleting and Renaming Existing Tables

You should delete database tables with caution, because it is extremely difficult to rebuild a database table that you delete by mistake. And, there is always a chance that some database application somewhere is still using the table. When you delete a table from the database, you delete the table structure from the data dictionary and remove all of the table's data from the database. To delete a table, you use the DROP TABLE command, which has the following syntax:

```
DROP TABLE tablename;
```

Restricted Actions	Restriction
Deleting a table from a user schema	Allowed only if the table does not contain any columns that other tables reference as foreign keys
Changing an existing column's data type	Allowed only if existing data in the column is compatible with the new data type. For example, you could change a VARCHAR2 data type to a CHAR data type, but you could not change a VARCHAR2 data type to a NUMBER data type.
Decreasing the width of an existing column	Allowed only if existing column values are NULL
Adding a primary key constraint to an existing column	Allowed only if current column values are unique (no duplicate values) and NOT NULL
Adding a foreign key constraint	Allowed only if current column values are NULL, or exist in the referenced table
Adding a UNIQUE constraint to a column	Allowed only if current column values are all unique
Adding a CHECK constraint	Allowed, but the constraint applies only to values users insert after the constraint is added
Changing a column's default value	Allowed, but the default value is only inserted for rows that are added after the change

Table 2-5 Restricted actions when modifying database tables

Recall that deleting a table is a restricted action, so the DROP TABLE command cannot succeed if the table represented by *tablename* contains columns that other tables reference as foreign keys. For example, you cannot drop the LOCATION table in the Northwoods University database, because the FACULTY table references the LOC_ID column as a foreign key.

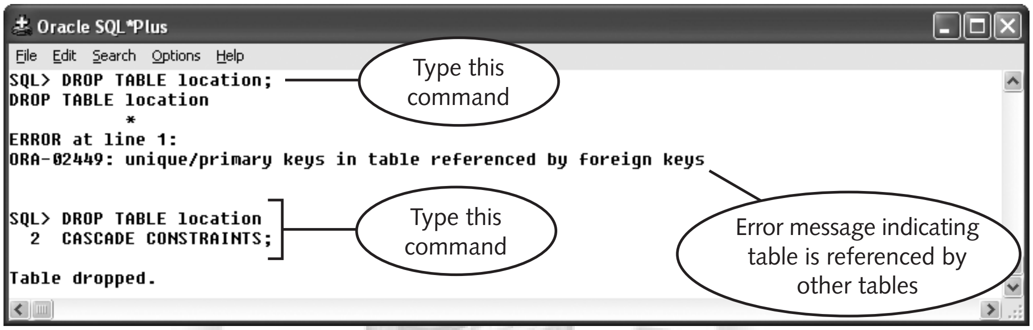
To drop a table that contains columns that other tables reference as foreign keys, you have two options. One option is to first drop all the tables that contain the foreign key references. For example, to delete the LOCATION table, you could first delete the FACULTY table, and then you could delete the LOCATION table. The second option is first to delete all of the foreign key constraints that reference the table to be deleted. To delete the foreign key constraints that reference a table, you use the CASCADE CONSTRAINTS option in the DROP TABLE command, which has the following syntax:

```
DROP TABLE tablename
CASCADE CONSTRAINTS;
```

When you execute the DROP TABLE command with the CASCADE CONSTRAINTS option, the system first drops all of the constraints associated with the table, and then drops the table. In the following set of steps, you try to drop the LOCATION table without the CASCADE CONSTRAINTS option, and note that an error occurs, because of the foreign key reference in the FACULTY table. You then drop the table using the CASCADE CONSTRAINTS option, which first deletes the table constraint.

To drop the LOCATION table, along with all of the constraints that reference the table, follow these steps:

1. Type **`DROP TABLE location;`** at the SQL prompt, and then press **Enter**. The error message "ORA-02449: unique/primary keys in table referenced by foreign keys" appears as shown in Figure 2-12, which indicates that one or more columns in the LOCATION table are referenced as foreign keys in other tables.


Oracle SQL*Plus
File Edit Search Options Help
Type this command
Type this command
Error message indicating table is referenced by other tables

```
SQL> DROP TABLE location;
DROP TABLE location
           *
ERROR at line 1:
ORA-02449: unique/primary keys in table referenced by foreign keys

SQL> DROP TABLE location
  2  CASCADE CONSTRAINTS;

Table dropped.
```

Figure 2-12 Commands to delete a database table

2. Type **`DROP TABLE location CASCADE CONSTRAINTS;`** at the SQL prompt, and then press **Enter**. The "Table dropped" message confirms that the DBMS successfully dropped the table.

To rename an existing table, you use the RENAME TO command, which has the following syntax:

```
RENAME old_tablename
TO new_tablename;
```

When you rename a table, the DBMS automatically transfers to the new table integrity constraints, indexes, and privileges that referenced the old table. However, when you rename a table, objects that referenced the old table, such as views and stored procedures and functions, become invalid. (You learn about views in Chapter 3, and you learn how to create stored procedures and functions in Chapter 9.) For example, you use the following command to rename the Northwoods University FACULTY table to NW_FACULTY:

```
RENAME faculty TO nw_faculty;
```

Adding Columns to Existing Tables

Sometimes an organization's data requirements change, and you need to add new columns to existing database tables. Recall from Table 2-4 that adding a new column to a table is an unrestricted action, so you can easily add new table columns as needed. The basic syntax of the command to add a new column to a table is:

```
ALTER TABLE tablename
ADD(columnname data_declaration constraints);
```

In this syntax, *columnname* is the name of the new data column. *Data_declaration* defines the new column's data type and maximum size, and *constraints* defines any constraints you want to place on the new column, such as foreign key references, check conditions, or value constraints. Northwoods University wants a column added to the FACULTY table that specifies each faculty member's employment start date. Next, you add the START_DATE column to the FACULTY table.

To add the START_DATE column to the FACULTY table:

1. Type the following command at the SQL prompt:

   ```
   ALTER TABLE faculty
   ADD (start_date DATE);
   ```

2. Press **Enter** to execute the command. The "Table altered" confirmation message appears and confirms that the DBMS successfully altered the table.

Modifying Existing Column Data Definitions

Sometimes you need to modify the data definition for an existing column. Recall from Table 2-5 that you can modify the data type of an existing column only if existing data values in the column rows are compatible with the new data type. As a result, you can only change a column's data type to another data type that stores the same kind of data. You can change VARCHAR2 columns to CHAR columns and vice versa, and you can change DATE columns to TIMESTAMP columns and vice versa. But, you *cannot* change VARCHAR2 columns to NUMBER columns or DATE columns, nor can you change NUMBER columns to VARCHAR2 or DATE columns.

You can also change the maximum size parameters for an existing column. Making an existing column's maximum size larger is an unrestricted action, and making an existing column's size smaller is a restricted action. You can make an existing column's size specification smaller only if the table does not contain any data. If the table contains rows, the values for the column being changed must be NULL.

The general syntax of the command to modify an existing column's data declaration is:

```
ALTER tablename
MODIFY(columnname new_data_declaration);
```

When you created the FACULTY table (see Figure 2-6), you defined the F_RANK column using the VARCHAR2 data type, with a maximum column width of eight characters. In the following set of steps, you modify the F_RANK column so it uses the CHAR data type, and has a maximum width of four characters.

To modify the data type and size of the F_RANK column:

1. At the SQL*Plus prompt, type the following command to modify the F_RANK column:

   ```
   ALTER TABLE faculty
   MODIFY (f_rank CHAR(4));
   ```

2. Press **Enter**. The "Table altered" message confirms that the DBMS successfully altered the table.

Deleting a Column

Sometimes you need to delete an existing column because the database applications no longer need the column's data. When the column is deleted, any data stored in that column is also removed from the database.

The general command to delete an existing column is:

```
ALTER TABLE tablename
DROP COLUMN columnname;
```

Renaming a Column

Suppose you need to change the name of the F_RANK column in the FACULTY table from F_RANK to FACULTY_RANK. To make this change, you could delete the existing F_RANK column, and then add to the table a new column named FACULTY_RANK. Of course if there were any data in the column, the data would be lost when you deleted the F_RANK column. Instead, you can issue the following ALTER TABLE command:

```
ALTER TABLE tablename
RENAME COLUMN old_columnname TO new_columnname;
```

To replace the F_RANK column with the FACULTY_RANK column:

1. At the SQL prompt, type **`ALTER TABLE faculty RENAME COLUMN f_rank TO faculty_rank;`**
2. Press **Enter** to execute the command. The "Table altered" message confirms that the DBMS successfully changed the name of the column.

Adding and Deleting Constraints

During the life cycle of a database system, the database applications change, and you may need to add or delete constraints in existing tables. For example, suppose after you create the Northwoods University FACULTY table, you need to add a UNIQUE constraint to the F_PIN column to ensure that each faculty member's PIN is unique. Table 2-5 shows that adding new constraints is a restricted action. It is difficult to add new constraints to tables that already contain data, because usually the data values do not conform to the new constraint. Deleting constraints is an unrestricted action, so it is easy to do.

To add a constraint to an existing table, you use the following command:

```
ALTER TABLE tablename
ADD CONSTRAINT constraint_name constraint_definition;
```

In this syntax, *constraint_name* is the name of the new constraint, based on the constraint naming convention. *Constraint_definition* specifies the constraint type (such as primary key or check condition), using the syntax you learned for creating constraints after all the column definitions are completed in the CREATE TABLE command. Next, you add the UNIQUE constraint to the FACULTY table to ensure that all F_PIN values are unique.

To add the UNIQUE constraint to the FACULTY table:

1. Type the following command at the SQL prompt:

   ```
   ALTER TABLE faculty
   ADD CONSTRAINT faculty_f_pin_uk UNIQUE (f_pin);
   ```

2. Press **Enter** to execute the command. The "Table altered" message confirms that the DBMS successfully added the constraint.

To remove an existing constraint, you use the following command:

```
ALTER TABLE tablename
DROP CONSTRAINT constraint_name;
```

In this syntax, *constraint_name* is the name you assigned to the constraint using the constraint naming convention. In this next set of steps you drop the UNIQUE constraint you just added to the F_PIN column in the FACULTY table.

To drop the constraint:

1. Type the following command at the SQL prompt:

   ```
   ALTER TABLE faculty
   DROP CONSTRAINT faculty_f_pin_uk;
   ```

2. Press **Enter** to execute the command. The "Table altered" message confirms that the DBMS successfully deleted the constraint.

Enabling and Disabling Constraints

Sometimes while you are developing new database applications, it is useful to disable constraints, then reenable the constraints when the application is finished. For example, suppose one programming team member is working on an application to add rows to the FACULTY table, while another team member is performing maintenance operations on the LOCATION table. (Recall that the LOC_ID column in the FACULTY table references the LOC_ID column in the LOCATION table as a foreign key.) If the team member working with the LOCATION table deletes all of the table rows, the team member working with the FACULTY table cannot insert any new rows, because there are no LOC_ID primary key values to reference.

When you create a new constraint using either the CREATE TABLE or ALTER TABLE commands, the DBMS immediately enables the constraint. When a constraint is **enabled**, the DBMS enforces the constraint when users attempt to add new data to the database. You can use SQL commands to disable constraints. You use the following command to disable an existing constraint:

```
ALTER TABLE tablename
DISABLE CONSTRAINT constraint_name;
```

Similarly, you use the following command to enable a constraint that you previously disabled:

```
ALTER TABLE tablename
ENABLE CONSTRAINT constraint_name;
```

If you try to disable a constraint that is currently disabled, or enable a constraint that is currently enabled, the DBMS executes the command anyway, and does not issue an error. Next you execute the commands to disable the FACULTY_LOC_ID_FK FOREIGN KEY constraint in the FACULTY table, and then reenable the constraint.

To disable and then reenable the constraint:

1. Type the following command at the SQL prompt, and then press **Enter**:

```
ALTER TABLE faculty
DISABLE CONSTRAINT faculty_loc_id_fk;
```

The "Table altered" message appears, which confirms that the DBMS successfully disabled the constraint.

2. Type the following command at the SQL prompt, and then press **Enter**:

```
ALTER TABLE faculty
ENABLE CONSTRAINT faculty_loc_id_fk;
```

   The "Table altered" message appears, which confirms that the DBMS successfully reenabled the constraint.

You are now familiar with how to create and modify Oracle10*g* database tables using SQL commands. Next you delete the FACULTY table, and then exit SQL*Plus.

To delete the FACULTY table and exit SQL*Plus:

1. Type the following command at the SQL prompt to delete the FACULTY table, and then press **Enter**:

```
DROP TABLE faculty CASCADE CONSTRAINTS;
```

2. To exit SQL*Plus, type **exit** at the SQL prompt, and then press **Enter**.
3. Save the Ch2Queries.sql file and close Notepad.

Chapter Summary

- Users interact with relational databases using high-level query languages that use English words such as SELECT, CREATE, ALTER, INSERT, and UPDATE. The standard query language for relational databases is Structured Query Language (SQL).
- SQL commands include data description language (DDL) commands, which create, modify, and delete database objects, and data manipulation language (DML) commands, which allow users to insert, update, delete, and view database data. Usually, only database administrators create and execute DDL commands.
- Each user account owns table and data objects in its own area of the database, which is called the user schema. Objects within a user schema are referred to as database objects or schema objects.
- When you create an Oracle10*g* database table, you specify the table name, the name of each data column, and the data type and size of each data column. Table names and column names must follow the Oracle naming standard.
- When you create a table, you define constraints, which define primary and foreign keys and restrict the data values that users can store in columns.
- Column data types ensure that users enter correct data values. Data types also allow the DBMS to use storage space more efficiently by optimizing how specific types of data are stored.
- Oracle10*g* stores character data using the VARCHAR2 and NVARCHAR2 data types, which store variable-length character data, and the CHAR and NCHAR data types, which store fixed-length character data.
- Oracle10*g* stores numerical data in columns that have the NUMBER data type. The precision of a NUMBER data type declaration specifies the total number of digits both to the left and to the right of the decimal point, and the scale specifies the number of digits to the right of the decimal point. Based on the precision and scale values, NUMBER columns can store integer, fixed-point, and floating-point values.
- Oracle10*g* uses the DATE and TIMESTAMP data types to store date and time data, and the INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND data types to store time intervals.
- Oracle10*g* uses the BLOB, BFILE, CLOB, and NCLOB data types to store binary data or large volumes of text data.
- Constraints restrict the data values that users can enter into database columns. Integrity constraints define primary key and foreign key columns; value constraints specify specific data values or ranges of values, whether the values can be NULL, whether the values have a default value, and whether a value must be unique. Table constraints restrict data values with respect to all other values in the table, and column constraints limit values for a single column, regardless of values in other rows.

- You can define constraints at the end of the CREATE TABLE command after all the columns are declared, or you can define each constraint within the column definition. Each constraint in a user schema must have a unique constraint name.
- SQL*Plus commands are not case sensitive, and the SQL*Plus interpreter ignores spaces and line breaks for formatting. You can create a SQL*Plus command using a text editor, copy the command, and then paste it into SQL*Plus. If the command has an error, you can edit the command in the text editor, and then copy, paste, and execute the command again.
- When SQL commands have errors, the SQL*Plus interpreter reports the line number that is causing the error, shows the position of the character that is causing the error, and returns an error code and description. When you need more information about an error code, you can look up the error code on the Oracle Technology Network Web site.
- You can use the DESCRIBE command to display a table's column names, data types, and NOT NULL constraints. You can use the data dictionary views to retrieve information about your database table names and constraint names.
- You can drop and rename tables, add new columns to tables, increase column widths, or drop existing constraints with no restrictions. You can modify a column data type or constraint only if the data that was inserted in the column is compatible with the updated data type or constraint. You can delete and rename table columns.
- You can disable constraints while you develop new database applications. When you disable a constraint, the DBMS does not enforce the constraint when you insert new data values.

REVIEW QUESTIONS

1. Which command is used to display the structure of a table in SQL*Plus?
2. What is the purpose of a value constraint?
3. You can use the ________________ data type to make certain that if the name John is stored in a column, it is stored using a total of nine spaces, rather than just four spaces.
4. The DATE data type uses the default format of DD-MON-YY. True or False?
5. All table and column names must start with a(n) ________________.
6. What command and option should be entered to delete a table that is referenced by a foreign key constraint in another table?
7. What is the default value assigned to a VARCHAR2 data type if no maximum size is stated in the column declaration?

8. All integrity constraints are displayed in the column list when you display the structure of a table. True or False?
9. Which view can be referenced to obtain a list of all objects that currently exist in the database?
10. By including a DEFAULT constraint for a column, you ensure that the column never contains a NULL value. True or False?

Multiple Choice

1. Which of the following constraints permit NULL values?
   a. Unique
   b. primary key
   c. NOT NULL
   d. none of the above
2. When you create an object, it is stored in your ________________.
   a. user table
   b. user schema
   c. data dictionary
   d. object list
3. Which of the following declares that the column can store a total of five digits, two of which are decimal values?
   a. NUMBER(3,2)
   b. NUMBER(5,2)
   c. NUMBER(float)
   d. NUMBER(p, 2)
4. Which of the following is a valid declaration for a DATE datatype?
   a. dobDATE(8)
   b. dobDATE(7)
   c. dobDATE
   d. dobDATE(DD-MON-YY)
5. Which of the following constraints must be included or the table cannot be created in Oracle10*g*?
   a. primary key
   b. foreign key
   c. NOT NULL
   d. none of the above

6. Which of the following commands should be used to change the name of a database table?
   a. RENAME
   b. ALTER TABLE
   c. MODIFY TABLE
   d. CHANGE
7. Which of the following commands should be used to change the name of a column?
   a. RENAME
   b. ALTER TABLE
   c. MODIFY TABLE
   d. CHANGE
8. When modifying a database table, which of the following is a restricted action?
   a. renaming a table
   b. deleting a column
   c. adding a constraint
   d. deleting a constraint
9. Which view can be used to determine the name of the database tables currently stored in the database?
   a. USERS
   b. CONSTRAINTS
   c. TAB_COLUMNS
   d. TABLES
10. Which of the following ALTER TABLE clauses is used to remove a column from a database table?
   a. MODIFY
   b. DELETE
   c. REMOVE
   d. none of the above

Problem-solving Cases

Figures 1-24 and 1-25 display the data and relationships within the Clearwater Traders database. Reference these figures when completing the following cases.

1. Based on the description of the Clearwater Traders database, determine the appropriate constraints for each table. Make certain you consider the following constraint types for each table in the database:
   - Primary key
   - Foreign key
   - Check condition
   - NOT NULL
   - Unique

   When drafting your results, list each table name, the constraint type, and the column that is referenced by the constraint.
2. Based on the sample data provided, determine the necessary column definition for each column contained in the following tables:
   - CUSTOMER
   - ORDERS
   - ORDER_LINE
   - ORDER_SOURCE
   - CATEGORY
   - ITEM
   - INVENTORY
3. Using the results from Case 1 and Case 2, write the SQL commands necessary to create the CUSTOMER, ORDERS, ORDER_LINE, ORDER_SOURCE, CATEGORY, ITEM, and INVENTORY tables. Remember to include the appropriate constraints for these tables in your SQL statements. Enter these commands into a Notepad file named Ch2Case3.sql and save it in the Chapter02\Cases folder on your data disk. Do not execute the commands at this time.
4. Using the SQL commands created in Case 3, create the CUSTOMER, ORDERS, and ORDER_SOURCE tables using the SQL*Plus utility. After the tables are created, use the DESCRIBE command to verify that the columns have been correctly defined for each of these tables.
5. Use the SQL*Plus utility to perform the following tasks:

   a. Change the name of the ORDERS table to ORDER.

   b. Delete the C_MI column from the CUSTOMER table.

   c. Change the definition of the O_METHPMT column of the ORDER table so it is defined as a CHAR datatype that stores only two characters.

d. Use the USER_CONSTRAINTS view to display the name of all constraints for the ORDER table.

e. Delete the ORDER_SOURCE table. If you receive an error message, take the appropriate corrective action, and then remove the table.

NOTE

Recall that foreign key constraints can create problems when dropping a table that is referenced by the constraint.)

f. Delete the CUSTOMER and ORDER tables.