

CHAPTER 2

SQL: THE BASICS

CHAPTER OBJECTIVES

In this chapter, you will learn about:

- ✓ The SQL*Plus Environment Page 52
- ✓ The Anatomy of a SELECT Statement Page 67
- ✓ Editing a SQL Statement Page 79
- ✓ The WHERE Clause: Comparison
and Logical Operators Page 95
- ✓ The ORDER BY Clause Page 114

Now that you are familiar with the concepts of databases and schema diagrams, you are ready to start with hands-on exercises. You will learn the basics of SQL*Plus, the software tool that allows you to execute statements against the Oracle database. After you familiarize yourself with SQL*Plus, you will be ready to write SQL statements, or queries, to retrieve the data. SQL statements can range from very simple to highly complex; they can be a few words long or a few hundred words long. In this chapter, you begin by writing simple SQL statements, but you will be able to build longer, more complex SQL queries very quickly.

52 *Lab 2.1: The SQL*Plus Environment***LAB
2.1****L A B 2 . 1**

THE SQL*PLUS ENVIRONMENT

LAB OBJECTIVES

After this lab, you will be able to:

- ✓ Identify Oracle's Client/Server Software
- ✓ Login and Logout of SQL*Plus

Oracle software runs on many different operating systems and hardware environments. You can use the SQL*Plus software under three different architectural configurations: as a stand-alone machine, in a client-server setup, or as iSQL*Plus within a three-tier architecture. Another piece of Oracle software, called SQL*Net (Version 7), Net8 (Version 8), or Oracle Net (Version 9i and 10g), provides the required communication protocol to the server.

STAND-ALONE ENVIRONMENT

SQL*Plus may be run in a stand-alone environment, where both the SQL*Plus client software and the Oracle database software reside on the same physical machine. This is the case when you install both the Oracle database server and the SQL*Plus software on your individual computer.

CLIENT-SERVER

A common setup is a client-server environment, also referred to as two-tier architecture, where a client communicates with the server. In this type of environment, Oracle's SQL*Plus tool resides on a client computer such as a PC or Unix workstation; the Oracle RDBMS software resides on a server. Figure 2.1 shows such a client-server architecture.

The client sends SQL statements to the server, and the server responds back with the result set. The job of the database server involves listening and managing many clients' requests, because in this configuration there are often multiple client machines involved.

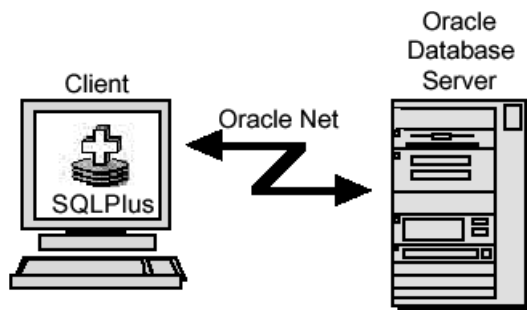


Figure 2.1 ■ Client-server architecture.

Instead of Oracle's SQL*Plus program, the client machine may run any other program with the ability to execute SQL statements against a database (e.g., Visual Basic or a custom-built Java program). For the client computer's programs to communicate with the Oracle database server, the individual client machine is typically configured with the Oracle Net software, or the client may establish an Open Database Connectivity (ODBC) connection.

THREE-TIER ARCHITECTURE

Starting with Oracle 8.1.7, you can use the *i*SQL*Plus interface in a Web browser to access the Oracle database. It performs the same actions as SQL*Plus. The advantage of *i*SQL*Plus is that you don't need to install and configure the SQL*Plus program or Oracle Net software on your client machine. As long as you use a compatible browser on your machine and know the URL of the Oracle HTTP server, you can access the database. As with any connection, you obviously need a valid user account and password.

Figure 2.2 shows the three-tiered architecture of an *i*SQL*Plus configuration. The first tier is the client's Web browser, and the middle tier is the Oracle HTTP server (Web server) that receives requests from the browser and forwards them via Oracle Net to the third tier, the Oracle database server. The Oracle Web server returns

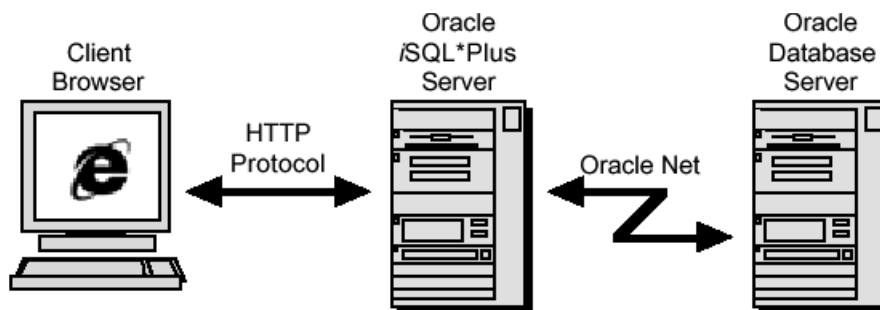


Figure 2.2 ■ Three-tier architecture.

54 Lab 2.1: The SQL*Plus Environment

LAB 2.1

results from the database server back to the Web browser for display. The three tiers may be on one machine but are typically on three different ones.

SQL AND THE ORACLE DATABASE SERVER

In the midst of all this software lies the SQL language. SQL commands are sent from the client software, also known as the *front end*, to the server, or *back end*. These commands send instructions to the server to tell it what services to provide. The server responds by sending back a result to the client, where it is displayed by the client software. Figure 2.3 shows a SQL statement that queries the DESCRIPTION column of the COURSE table. The SQL statement is sent to the Oracle server and the result is displayed by SQL*Plus.

USER ID AND PASSWORD

To connect to the database and communicate via SQL*Plus, you must have a user ID that has been created for you. For the purposes of all examples in this book, you use the user name *STUDENT* and the password *LEARN*. Note that the user ID and password are not case sensitive.



If you have not yet created the STUDENT schema according to the instructions on the companion Web site located at <http://authors.phptr.com/rischert3e>, you will not be able to log in with the STUDENT user ID and the LEARN password. You may want to continue to read through this lab first, create the STUDENT schema, and then perform the exercises in this lab.

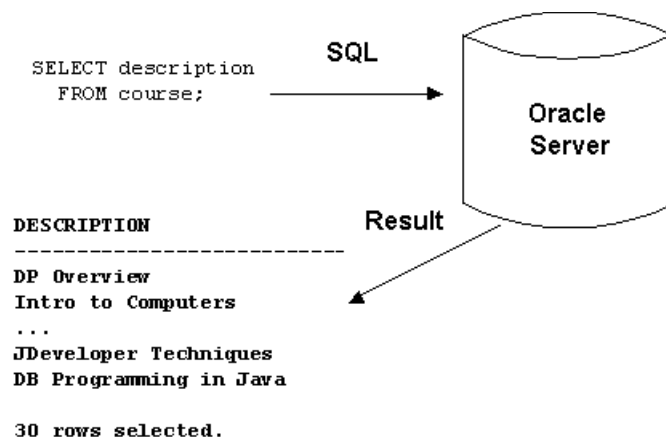


Figure 2.3 ■ SQL and the Oracle database server.

ACCESSING THE ORACLE DATABASE SERVER

LAB 2.1

You can access the Oracle server through various front-end tools. This book will discuss the use of Oracle's own SQL*Plus software (available as a graphical Windows environment and as a command line interface) and the browser-based iSQL*Plus.

This lab will teach you some of the basics of SQL*Plus, as this tool is almost always found in any Oracle database environment. The log on screens for SQL*Plus and the browser-based iSQL*Plus are slightly different, but easily understood. You can use either SQL*Plus or iSQL*Plus to execute your SQL statements, or perhaps you chose another front-end query tool that also allows you to enter SQL commands. (The companion Web site to this book lists other alternative query tools.) Differences between SQL*Plus or iSQL*Plus are pointed out to you as you work through the book. You can assume that with very few exceptions the functionality of iSQL*Plus and SQL*Plus are very similar, if not identical.



*When working through this book, you have a choice to use either a browser and access iSQL*Plus or use the SQL*Plus software installed on your machine.*

SQL*PLUS CLIENT FOR WINDOWS

If the SQL*Plus program is installed on your Windows machine, you can access it by choosing Programs, then Oracle, Application Development, and SQL Plus. This launches the program and displays the Log On dialog box similar to Figure 2.4. Enter as the User Name STUDENT and as the Password LEARN.

If your database is installed on the same machine as your SQL*Plus client, you don't need to enter a value in the Host String field. If you are connecting to a

The image shows a Windows-style dialog box titled "Log On". It contains three text input fields stacked vertically. The first field is labeled "User Name:" with a small icon to its left. The second field is labeled "Password:" with a small icon to its left. The third field is labeled "Host String:" with a small icon to its left. Below the input fields are two buttons: "OK" on the left and "Cancel" on the right.

Figure 2.4 ■ Windows graphical user interface log on dialog box.

56 Lab 2.1: The SQL*Plus Environment

LAB 2.1

remote Oracle database, enter the Oracle Net connection string supplied to you by your Oracle database administrator and recorded in your TNSNAMES.ORA file. You will learn more about this special file later.

Figure 2.5 shows how your screen looks once you have successfully connected to the server. Effectively, you have established a connection with the Oracle database as the user STUDENT. The client and the server may now communicate with each other.

When you see the SQL> command prompt, SQL*Plus is ready to accept your commands and you may begin to type. This is the default prompt for SQL*Plus.

To log out, either type EXIT or QUIT and press enter. Alternatively, you can choose Exit from the File menu or simply use your mouse to close the window.

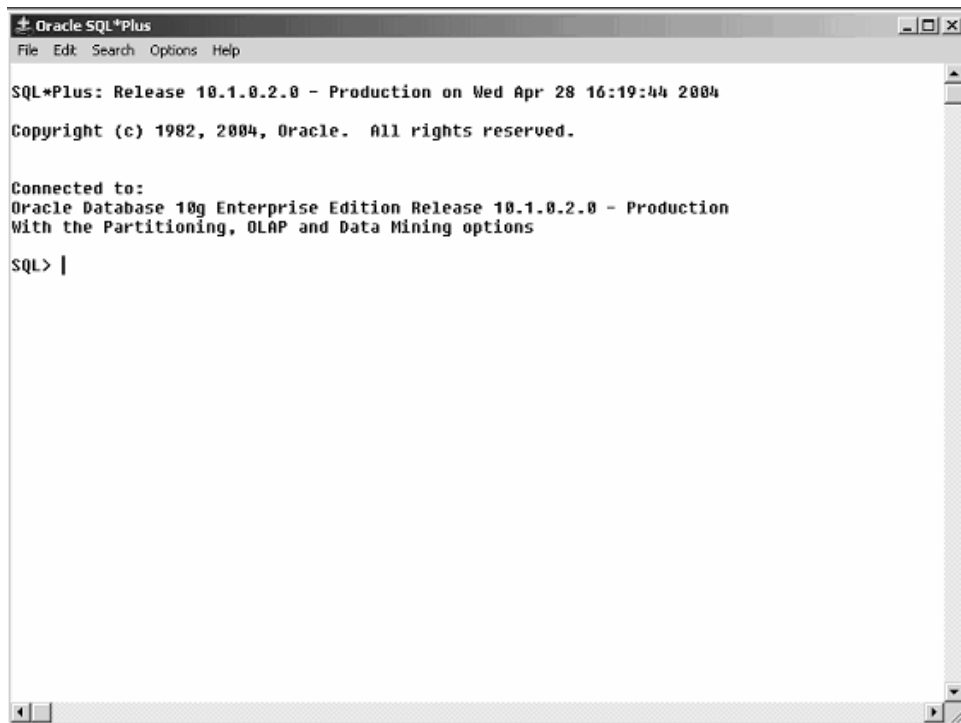


Figure 2.5 ■ SQL*Plus prompt.

CONNECTING WITH A WEB BROWSER: *i*SQL*PLUS

Instead of the SQL*Plus software program, you can also use the Web-based version called *i*SQL*Plus. To access the *i*SQL*Plus interface through your Web browser, you enter a URL. A Log on dialog similar to Figure 2.6 will appear. Here, the URL

is `http://scooby:5560/isqlplus` and will obviously be different for your individual installation.

A valid URL to connect to *iSQL*Plus* is in the form of `http://machine_name.domain:port/isqlplus`. For example, `http://mymachine.acme.com:5560/isqlplus` is an example of a URL format. As part of the default Oracle installation, you will usually see the *iSQL*Plus* port number displayed. If you are unsure about your specific port number, try the default port 5560.

Also notice in Figure 2.6 that the domain is not shown, only the machine name `scooby`. Because the machine is on a local network, you can omit the domain. Instead of the name of the machine, you can also enter the IP address. If your Oracle database server is on your own machine and you want to access *iSQL*Plus*, you can substitute `localhost` instead and your URL will read `http://localhost:5560/isqlplus`. Alternatively, you can use the IP address of `127.0.0.1`.

Enter the user ID and password in the appropriate boxes. You don't need to supply the Connection Identifier (also called Host string) to connect to the default database instance.

Figure 2.7 displays the screen you see once you have successfully logged in. Notice the *iSQL*Plus* Workspace and the message "Connected as `STUDENT@orcl`" on the upper right-hand side of the screen. This indicates the name of the login user,

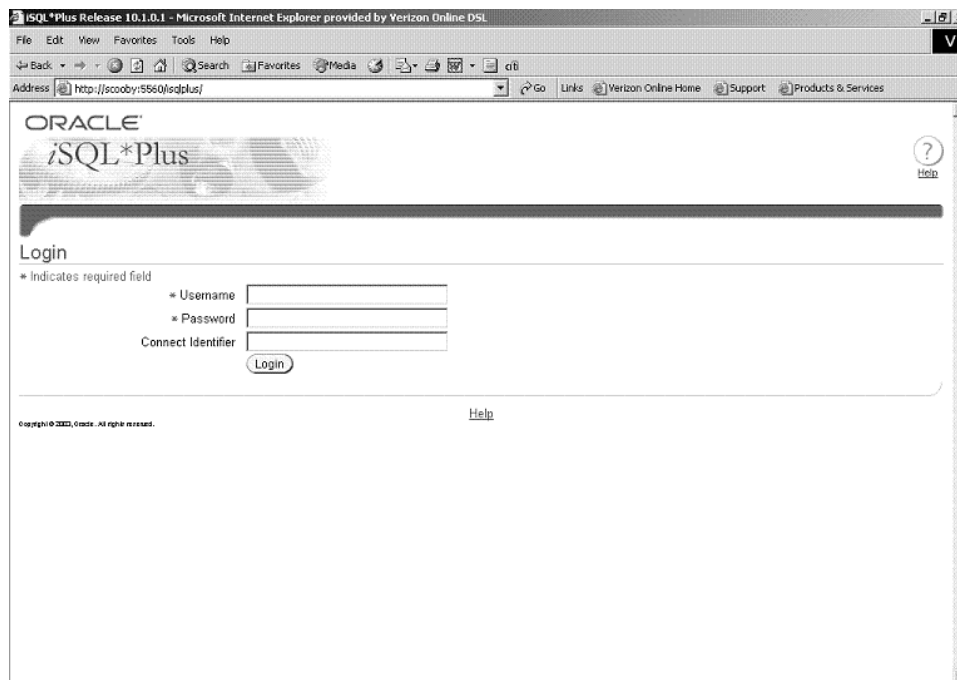


Figure 2.6 ■ *iSQL*Plus* login screen.

58 Lab 2.1: The SQL*Plus Environment

LAB 2.1



Figure 2.7 ■ iSQL*Plus Workspace.

which is STUDENT, and the name of the Oracle database instance you are connected to, called ORCL.

At the Enter statements text box, also referred to as the input area, you can enter commands. If you want to run a script (e.g., the script you need to execute to generate the STUDENT schema), you can enter the path and name of the script or click the Load Script button to locate the script. Once the script is loaded into the input area, you can edit the script or simply click the Execute button to execute the script. To logout and return to the Login screen, click on the Logout icon.

STARTING THE iSQL*PLUS APPLICATION SERVER

For the Windows environment, the iSQL*Plus application server is installed as a Windows service as part of the default Oracle database server installation and usually started automatically.

For other operating systems—or if you prefer to start iSQL*Plus from the command prompt—use the following syntax `%oracle_home%\bin\isqlplusctl start`. For example, if `C:\ORACLE\ORA10` is your Windows Oracle home directory where the files for the Oracle database and application server are installed, you start the iSQL*Plus application server with this command `C:\oracle\ora10\bin\isqlplusctl start`.

Refer to the companion Web site for more information on general *iSQL*Plus* installation and configuration questions.

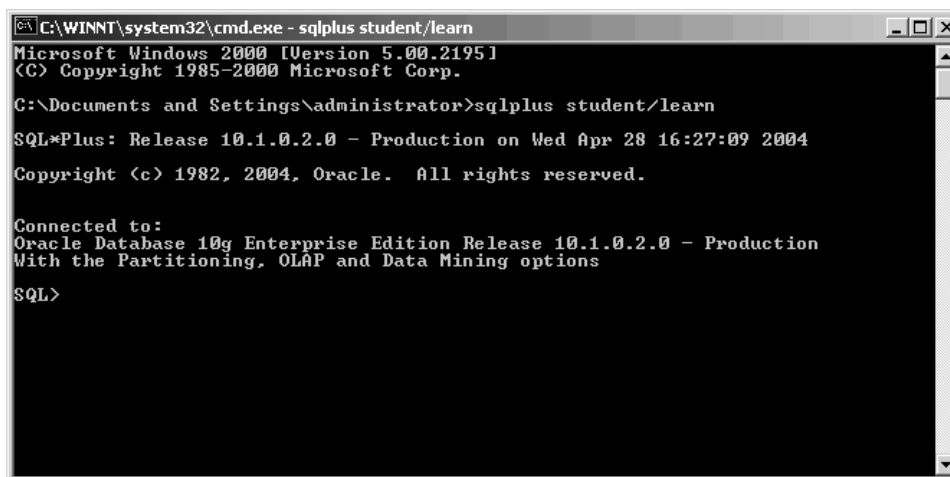
COMMAND-LINE INTERFACES FOR SQL*PLUS

In place of a graphical user interface such as SQL*Plus for Windows or *iSQL*Plus*, you may use a command-line interface. A command-line interface is available with every Oracle version. Frequently, you will use this interface in operating systems such as Linux or Unix. Even Windows has a command-line interface and you will see it displayed in Figure 2.8. All SQL*Plus and SQL commands operate for this interface just the same. Note that depending on the operating system, your editor, as well as the cut and paste commands, may be different.

To invoke SQL*Plus, you type `sqlplus` at the respective operating system's command prompt. For Windows you start SQL*Plus by typing `C:\> sqlplus` from the Windows command prompt. In this example, the username and password is supplied to start SQL*Plus. You can also enter `sqlplus` and you will be prompted for the user name and password or `sqlplus student`, which will prompt for the password.

THE REMOTE DATABASE AND COMMON LOG-ON PROBLEMS

Often the database resides on a machine other than your client machine, or you have a choice of accessing different databases. In these cases you need to supply the name of the database in the Host String box of the Log On dialog box (see Figure 2.9) or the Connection Identifier box in *iSQL*Plus*. For example, to connect to a database called *ITCHY* you have enter this name in the Host String box.



```
C:\WINNT\system32\cmd.exe - sqlplus student/learn
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Documents and Settings\administrator>sqlplus student/learn
SQL*Plus: Release 10.1.0.2.0 - Production on Wed Apr 28 16:27:09 2004
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 2.8 ■ Command line-based SQL*Plus under the Windows operating system.

60 Lab 2.1: The SQL*Plus Environment

LAB 2.1

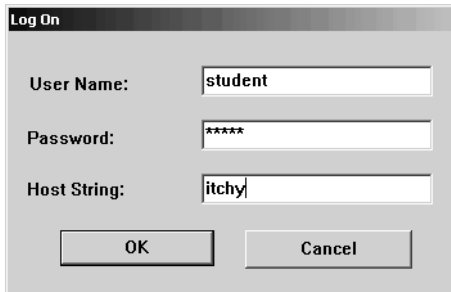


Figure 2.9 ■ SQL*Plus Windows graphical user interface log on dialog box.

The host string matches an entry in a file called TNSNAMES.ORA, which lists the database's IP address (or the machine name) and database instance name.

Essentially, the TNSNAMES.ORA file is a file containing a list of databases with their respective technical connection information. Your database administrator can help you with the configuration and setup of this file if you have a remote database setup.

Following is an excerpt of a TNSNAMES.ORA file. The entries in your file will obviously vary. If you supply the host string ITCHY at log in, SQL*Plus will look up the ITCHY entry in the TNSNAMES.ORA file. The HOST entry shows the IP address (if you use a TCP/IP network), which is listed as 169.254.147.245. Alternatively, you can enter the machine name. The SID entry identifies the name of the Oracle instance; here the instance is called ORCL. (When you install Oracle with the default options, you will be asked to supply such an instance name [SID]. A common default name is ORCL.)

```
ITCHY =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS =
        (PROTOCOL = TCP)
        (Host = 169.254.147.245)
        (Port = 1521)
      )
    )
    (CONNECT_DATA = (SID = ORCL)
  )
)
SCRATCHY =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP) (HOST = milly.columbia.edu) (PORT = 1521))
    )
    (CONNECT_DATA =
```

Lab 2.1: The SQL*Plus Environment 61

LAB 2.1

```
(SERVER = DEDICATED)
(SERVICE_NAME = scraty.columbia.edu)
)
)
```

Your TNSNAMES.ORA file may contain an entry called DEFAULT. If you do not supply a Host String in the Log On dialog box, you will be connected to the database listed under the DEFAULT option. Note, depending on your individual setup, you may at times need to specify or omit the .WORLD suffix next to the host name (such as ITCHY.WORLD or simply ITCHY) in the TNSNAMES.ORA file. Additionally, Oracle 9i and 10g installations allow the use the format of the second entry called SCRATCHY. It uses a service name instead of the SID.

If you are using an Oracle 10g client such as SQL*Plus for Windows, you can use a new feature called easyconnect. It allows you to make a connection without the entry being present in the TNSNAMES.ORA file. For example, you can connect to SCRATCHY by using this connect identifier in the Host String box: milly.columbia.edu:1521/scraty.columbia.edu. It lists the machine name called MILLY.COLUMBIA.EDU followed by the port number (the default port of the Oracle database is typically 1521), followed by the service name SCRATY.COLUMBIA.EDU.

COMMON LOG-ON PROBLEMS

Although we cannot possibly list all the errors and solutions to all log-on problems, here are two very common Oracle error messages.

A TNS error usually deals with the connectivity between the server and the client. The following message is displayed if the connect identifier could not be resolved. This may be due to an invalid host string. Check the values and retry.

```
ORA-12154: TNS: could not resolve the connect identifier specified
```

The next error occurs if you entered the wrong password or user name when the Oracle server attempted to authenticate you as a valid user. Double-check the spelling of your user name, which is STUDENT, and password, which is LEARN. (If you cannot log on with this ID and password, check the readme.txt file regarding the installation of the STUDENT schema.)

```
ORA-01017: invalid username/password; logon denied
```

EXITING FROM SQL*PLUS OR iSQL*PLUS

There are a number of ways to exit SQL*Plus. You can type EXIT or select Exit from the File menu in the SQL*Plus Windows version. For iSQL*Plus, you click the Logout icon rather than typing EXIT as this will free up system resources. Exiting ends the session and the STUDENT user is no longer connected to the database.

62 Lab 2.1: The SQL*Plus Environment

LAB 2.1

However, there may be other client machines connected to the Oracle database; the server software continues to run, regardless of whether a client is connected to it.

CREATING THE STUDENT SCHEMA

Now that you know how to log on to the Oracle database using SQL*Plus or iSQL*Plus, this is a good time to read the readme.txt file you downloaded from the Web site located at <http://authors.phptr.com/rischert3e> and create the STUDENT schema if you have not already done so.

Unless specifically mentioned, we will not differentiate between SQL*Plus and iSQL*Plus commands because many are almost identical. For a list of unsupported commands see Appendix C, “SQL*Plus Command Reference.”



*All commands in SQL*Plus require the user to press the Enter key to execute them. In iSQL*Plus you always need to press the Execute button. The reminder to press the Enter key or the Execute button will not be included in the rest of the examples and exercises in this book.*

LAB 2.1 EXERCISES

2.1.1 IDENTIFY ORACLE'S CLIENT/SERVER SOFTWARE

- a) Identify which piece of Oracle software is the client, which is the server, and how they communicate with each other.
- b) What is the role of SQL between client and server?

2.1.2 LOGIN AND LOGOUT OF SQL*PLUS

- a) Once you have logged into SQL*Plus (not iSQL*Plus) with the user ID STUDENT and password LEARN, what information does the SQL*Plus screen show you? (If you do not have access to SQL*Plus, please answer the question by referring to Figure 2.5.)
- b) What do you learn when you type `DESCRIBE student` and press `Enter`? If you use iSQL*Plus, click the Execute button instead of pressing `Enter`.
- c) Execute the following command and describe what you see:
`SHOW ALL.`

LAB 2.1 EXERCISE ANSWERS

LAB 2.1

2.1.1 ANSWERS

- a) Identify which piece of Oracle software is the client, which is the server, and how they communicate with each other.

*Answer: SQL*Plus or the browser displaying iSQL*Plus is the client and the Oracle RDBMS is the server. In an Oracle 9i or 10g environment, Oracle Net is the protocol that facilitates the communications.*

- b) What is the role of SQL between client and server?

Answer: SQL commands are issued from the client, telling the server to perform specific actions. The server sends back the results of those instructions to the client software, where they are displayed.

2.1.2 ANSWERS

- a) Once you have logged into SQL*Plus (not iSQL*Plus) with the user ID STUDENT and password LEARN, what information does the SQL*Plus screen show you? (If you do not have access to SQL*Plus, please answer the question by referring to Figure 2.5.)

*Answer: The screen shows which version of SQL*Plus you are using, the current date and time, Oracle copyright information, and the version of the Oracle database you are connected to. After this information is displayed, you see the SQL> command prompt. At this prompt you are able to enter commands.*

PL/SQL is another Oracle language addressed in a separate book in this series *Oracle PL/SQL by Example* by Benjamin Rosenzweig and Elena Silvestrova (Prentice Hall, 2004).

- b) What do you learn when you type `DESCRIBE student` and press Enter? If you use iSQL*Plus, click the Execute button instead of pressing Enter.

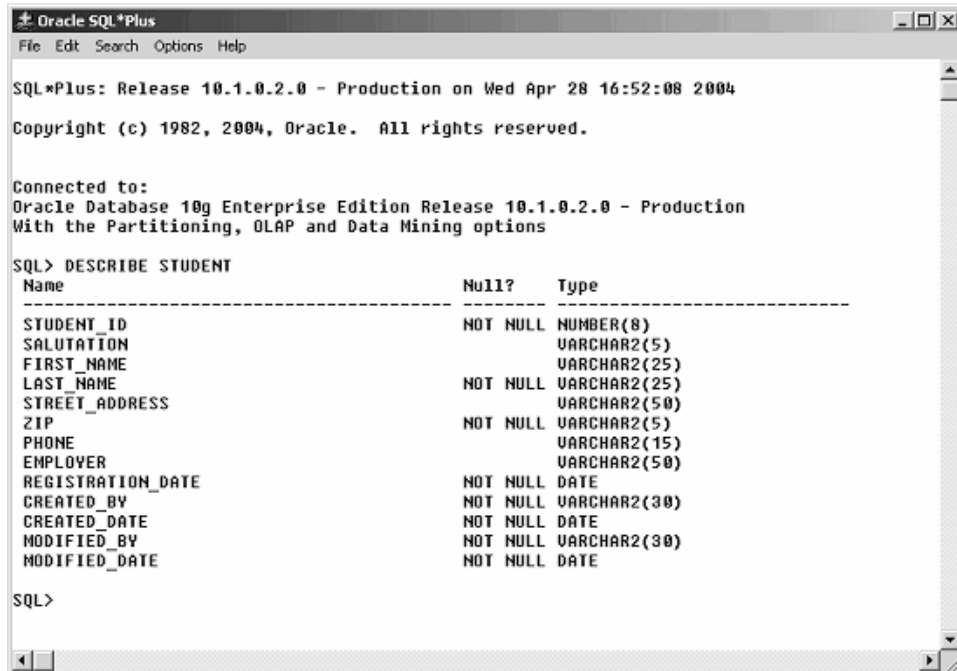
Answer: You find out about the structure of the STUDENT table, specifically its column names, whether those columns allow nulls and the datatype of each column.

To write SQL statements, you need to know a table's column names and their datatypes. The SQL*Plus DESCRIBE command displays this information and shows if a column does not allow null values.

Many SQL*Plus commands may be abbreviated. For instance, DESCRIBE may be shortened to DESC. Retype the command using this abbreviation and compare the results. Figure 2.10 displays the result of the DESCRIBE command executed in SQL*Plus.

64 Lab 2.1: The SQL*Plus Environment

LAB 2.1



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Apr 28 16:52:08 2004

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> DESCRIBE STUDENT
Name                                Null?    Type
-----
STUDENT_ID                          NOT NULL NUMBER(8)
SALUTATION                            VARCHA2(5)
FIRST_NAME                           VARCHA2(25)
LAST_NAME                             NOT NULL VARCHA2(25)
STREET_ADDRESS                       VARCHA2(50)
ZIP                                    NOT NULL VARCHA2(5)
PHONE                                  VARCHA2(15)
EMPLOYER                              VARCHA2(50)
REGISTRATION_DATE                     NOT NULL DATE
CREATED_BY                            NOT NULL VARCHA2(30)
CREATED_DATE                          NOT NULL DATE
MODIFIED_BY                           NOT NULL VARCHA2(30)
MODIFIED_DATE                         NOT NULL DATE

SQL>

```

Figure 2.10 ■ Executing the SQL*Plus DESCRIBE command.



*SQL*Plus is not case sensitive; the user ID, password, and SQL*Plus commands may all be entered in either upper or lowercase, or a combination of the two. Throughout this book, they are in uppercase for easy identification. In the next lab you will learn about formatting your SQL statements and when it is appropriate to capitalize words.*

COMMON DATATYPES

Every column in Oracle must have a datatype, which determines what type of data can be stored.

DATE

The DATE datatype is used to store date and time information. By default the display format for a date is DD-MON-YY. For example, July 4, 2003 displays as 04-JUL-03. There are a number of functions you can use to change the display format or to show the time, which you will learn about in “Chapter 4, Date and Conversion Functions.”

NUMBER

Columns with the datatype NUMBER only allow numerical data; no text, hyphens, or dashes are allowed. A column defined as NUMBER(5,2) can have a maximum of three digits before the decimal point and two digits after the decimal point. The first digit (5) is called the *precision*; the second digit (2) is referred to as the *scale*. The smallest allowed number is -999.99 and the largest is 999.99. A column definition with a zero scale such as NUMBER(5) or NUMBER(5,0) allows integers in the range from -99,999 to 99,999.

VARCHAR2 AND CHAR

The VARCHAR2 and CHAR datatypes store alphanumeric data (e.g., text, numbers, special characters, etc.). VARCHAR2 is the variable length datatype and the most commonly used alphanumeric datatype; its maximum size is 4,000 characters. The main difference between VARCHAR2 and CHAR is that the CHAR datatype is a fixed-length datatype and any unused room is blank padded with spaces.

For example, a column defined as CHAR(10) and containing the four-character length value of JOHN in a row will have six blank characters padded at the end to make the total length 10 spaces. (If the column is stored in a VARCHAR2(10) column instead, it stores four characters only.) A CHAR column can store up to 2,000 characters.

The CLOB database allows you to store large amounts of textual data. It replaces the LONG datatype, which is desupported in Oracle 10g.

OTHER

Datatypes such as BFILE or BLOB require access through specific purpose functions in very highly specialized ways that go beyond the objectives of this book. In addition to the datatypes mentioned, Oracle also has additional datatypes to support specific national character sets (e.g., NCLOB, NVARCHAR2), intermedia datatypes, and spatial (geographic) data. Oracle also gives you the ability to create your own customized object datatypes.

Refer to Appendix I, "Oracle Datatypes," for a detailed listing of the various datatypes. For most SQL operations, you typically use the NUMBER, VARCHAR2, and various DATE-related datatypes. They are the most commonly used datatypes where the vast majority of data is stored.

c) Execute the following command and describe what you see: `SHOW ALL`.

*Answer: You will see a list of SQL*Plus environmental variables and their current settings. Using the SET command, many of them can be changed to suit your needs for a SQL*Plus session, which is defined as the time in between when you log in and log out of SQL*Plus. When you start your next SQL*Plus session, however, all commands will be set back to their defaults.*

66 Lab 2.1: The SQL*Plus Environment

LAB 2.1

It is important to note here that SQL*Plus commands, such as SHOW and DESCRIBE, are *not* part of the SQL language. You will begin to type SQL commands using the SQL*Plus tool in the next lab.

If you use iSQL*Plus, you can change the environment variables and settings by clicking the Preferences icon.

LAB 2.1 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

- 1) The DESC command displays column names of a table.
 - a) True
 - b) False
- 2) Anyone can connect to an Oracle database as long as he or she has the SQL*Plus software.
 - a) True
 - b) False
- 3) The SQL*Plus command `SHOW USER` displays your login name.
 - a) True
 - b) False
- 4) Typing `SHOW RELEASE` at the prompt displays the version number of SQL*Plus you are using.
 - a) True
 - b) False
- 5) The COST column of the COURSE table is defined as NUMBER(9,2). The maximum cost of an individual course is 9,999,999.99.
 - a) True
 - b) False

Answers appear in Appendix A, Section 2.1.

L A B 2 . 2

THE ANATOMY OF A SELECT STATEMENT

**LAB
2.2**

LAB OBJECTIVES

- After this lab, you will be able to:
- ✓ Write a SQL SELECT Statement
 - ✓ Use DISTINCT in a SQL Statement

THE SELECT STATEMENT

When you write a SQL query, it is usually to answer a question such as “How many students live in New York?” or “Where, and at what time, does the Unix class meet?” A SQL *SELECT statement*, or *SQL query*, is used to answer these questions. A SELECT statement can be broken down into a minimum of two parts: the *SELECT list* and the *FROM clause*. The SELECT list usually consists of the column or columns of a table(s) from which you want to display data. The FROM clause states on what table or tables this column or columns are found. Later in this chapter, you will learn some of the other clauses that can be used in a SELECT statement.

HOW DO YOU WRITE A SQL QUERY?

Before formulating the SELECT statement, you must first determine the table where the information is located. A study of the schema diagram reveals that the COURSE table provides descriptions of courses. (You can also refer to Appendix E, “Table and Column Descriptions.”)

The following SELECT statement provides a list of course descriptions:

68 Lab 2.2: The Anatomy of a SELECT Statement

```
SELECT description
FROM course
```

The SELECT list shows the single column called DESCRIPTION, which contains this information. The DESCRIPTION column is found on the COURSE table as specified in the FROM clause. When the statement is executed, the result set is a list of all the values found in the DESCRIPTION column of the COURSE table:

**LAB
2.2**

```
DESCRIPTION
-----
DP Overview
Intro to Computers
...
JDeveloper Techniques
DB Programming in Java

30 rows selected.
```



Many of the result sets displayed throughout this book do not list all the rows. This is denoted with a line of “...” in the middle of the output. Typically, you will see the beginning and the ending rows of the result set and the number of rows returned. The resulting output of the SQL command is displayed in a bold font to easily distinguish the output from the commands you enter.

EXECUTING THE SQL STATEMENT

SQL*Plus does not require a new line for each clause, but it requires the use of a semicolon (;) at the end of each SQL statement to execute it. (Figure 2.11 shows the result of the execution of the previously mentioned SQL query in SQL*Plus.) Alternatively, the forward slash (/) may be used on a separate line to accomplish the same. In *iSQL*Plus* a semicolon or forward slash is not required, you only need to press the Execute button:

```
SQL> SELECT description
      2 FROM course;
```

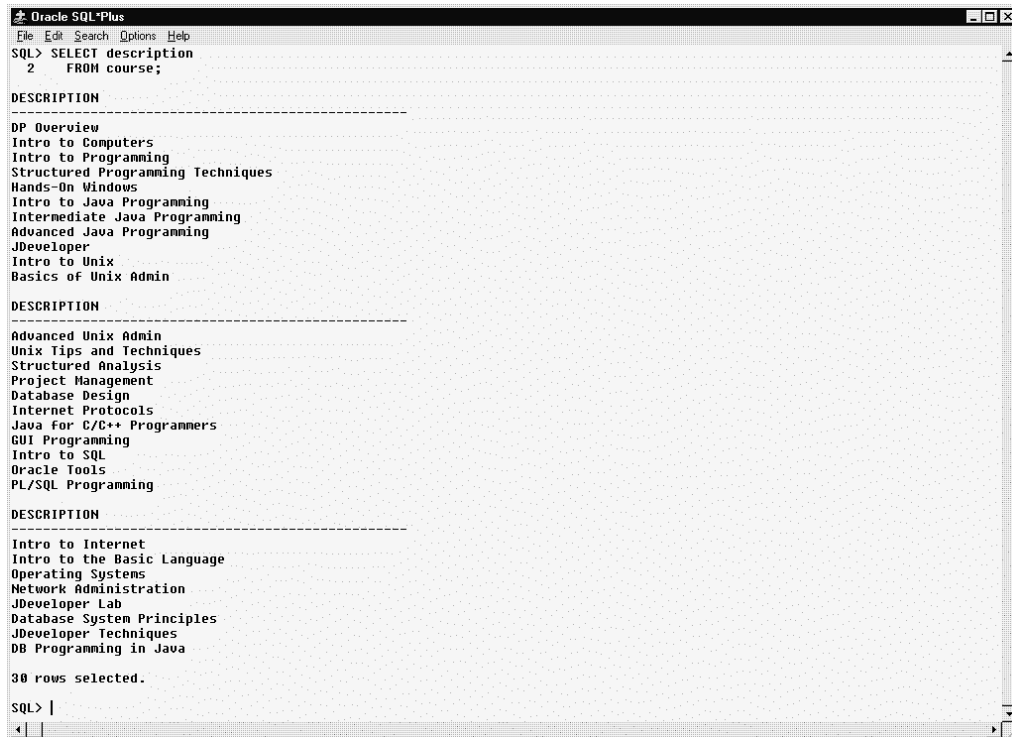
Or:

```
SQL> SELECT description
      2 FROM course
      3 /
```



*The SQL*Plus commands such as DESC or SHOW USER discussed in the previous lab are not SQL commands and therefore do not require a semicolon or forward slash.*

Lab 2.2: The Anatomy of a SELECT Statement 69



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT description
2 FROM course;

DESCRIPTION
-----
DP Overview
Intro to Computers
Intro to Programming
Structured Programming Techniques
Hands-On Windows
Intro to Java Programming
Intermediate Java Programming
Advanced Java Programming
JDeveloper
Intro to Unix
Basics of Unix Admin

DESCRIPTION
-----
Advanced Unix Admin
Unix Tips and Techniques
Structured Analysis
Project Management
Database Design
Internet Protocols
Java for C/C++ Programmers
GUI Programming
Intro to SQL
Oracle Tools
PL/SQL Programming

DESCRIPTION
-----
Intro to Internet
Intro to the Basic Language
Operating Systems
Network Administration
JDeveloper Lab
Database System Principles
JDeveloper Techniques
DB Programming in Java

30 rows selected.

SQL> |

```

Figure 2.11 ■ Executing the SELECT statement in SQL*Plus.

LAB
2.2

RETRIEVING MULTIPLE COLUMNS

To retrieve a list of course descriptions and the cost of each course, include the COST column in the SELECT list:

```

SELECT description, cost
FROM course

DESCRIPTION                COST
-----
DP Overview                 1195
Intro to Computers          1195
...
JDeveloper Techniques      1195
DB Programming in Java

30 rows selected.

```

When you want to display more than one column in the SELECT list, separate the columns with commas. It is good practice to include a space after the comma for readability. The order of columns in a SELECT list will determine the order in which the columns are displayed in the output.

70 *Lab 2.2: The Anatomy of a SELECT Statement***SELECTING ALL COLUMNS****LAB
2.2**

You can also select all columns in a table with the asterisk (*) wildcard character. This is handy so you don't have to type all columns in the SELECT list. The columns are displayed in the order in which they are defined in the table. This is the same order you see when you use the SQL*Plus DESCRIBE command. If you execute this command you will notice that the columns wrap in SQL*Plus (not iSQL*Plus) as there is not sufficient room to display them in one line. You will learn how to format the output shortly.

```
SELECT *
FROM course
```

ELIMINATING DUPLICATES WITH DISTINCT

The use of DISTINCT in the SELECT list eliminates duplicate data in the result set. The following SELECT statement retrieves the last name and the corresponding zip code for all rows of the INSTRUCTOR table.

```
SELECT last_name, zip
FROM instructor
```

LAST_NAME	ZIP
-----	----
Hanks	10015
Wojick	10025
Schorin	10025
Pertez	10035
Morris	10015
Smythe	10025
Chow	10015
Lowry	10025
Frantzen	10005
Willig	

10 rows selected.

Notice that there are 10 rows, yet only nine instructors have zip codes. Instructor Willig has a NULL value in the ZIP column. If you want to show only the distinct zip codes of the table, you write the following SELECT statement. The last row shows the NULL value.

```
SELECT DISTINCT zip
FROM instructor
```

ZIP

10005
10015

Lab 2.2: The Anatomy of a SELECT Statement 71

10025

10035

5 rows selected.



By definition, a NULL is an unknown value, and a NULL does not equal another NULL. However, there are exceptions: If you write a SQL query using DISTINCT, SQL will consider a NULL value equal to another NULL value.

LAB
2.2

From Chapter 1, “SQL and Data,” you already know that a primary key is always unique or distinct. Therefore, the use of DISTINCT in a SELECT list containing the primary key column(s) is unnecessary. The ZIP column in the INSTRUCTOR table is not the primary key and can therefore contain duplicate values.

DISPLAYING THE NUMBER OF ROWS RETURNED

You may notice that SQL*Plus sometimes does not show the number of rows returned by the query, but rather depends on the feedback settings for your SQL*Plus session. Typically, the feedback is set to 6 or more rows. In the previous example the feedback was set to 1, which displays the feedback line even when there is only one row returned. You will find this setting useful if your result set returns less than the default six rows and if any of the rows return nulls, which display as a blank. Otherwise, you may think it is not a row or value. To display the exact number of rows returned until you exit SQL*Plus, enter the SQL*Plus command:

```
SET FEEDBACK 1
```

To display your current settings use the SHOW ALL command or simply SHOW FEEDBACK. (If you want to retain certain SQL*Plus settings, you can create a login.sql file for your individual computer in a client-server setup. You can also create a glogin.sql file for all users if you want all to have the identical settings or if you use iSQL*Plus. See the companion Web site for more information.)

SQL STATEMENT FORMATTING CONVENTIONS

You will notice that the SQL statements presented in this and all other books in this series follow a common format. The use of uppercase for SELECT, FROM, and other Oracle keywords is for emphasis only, and distinguishes them from table and column names, which you see in the SQL statement as lowercase letters. A standard format enhances the clarity and readability of your SQL statements and helps you detect errors more easily. Refer to Appendix B, “SQL Formatting Guide,” for the formatting guidelines used throughout.

72 Lab 2.2: The Anatomy of a SELECT Statement

CANCELLING A COMMAND AND PAUSING THE OUTPUT

If you want to stop a command while the statement is still executing, you can press CTRL+C in SQL*Plus for Windows or the Cancel button in *iSQL*Plus*.

LAB 2.2

If your result in SQL*Plus is fairly large, you can examine the output by scrolling up and down. If you wish to look at the rows one screen at a time, use the SQL*Plus SET PAUSE ON command. This command displays one screen at a time and to change the number of lines displayed per screen to use the SET PAGESIZE *n* command where *n* is the number of rows per page. To continue to the next screen, press the Enter key in SQL*Plus. If you want to stop scrolling through the screens and return to the SQL> prompt, press CTRL + C. Remember to issue the SET PAUSE OFF command to stop the feature when you are done!

In *iSQL*Plus* you can choose to display only a specific number of rows per page by clicking on Preferences, Interface Configuration, Output Page Setup, and then Multiple Pages. If the output has more than the specified number of rows, you will see a Next Page button that lets you move to the next page of rows.

LAB 2.2 EXERCISES

2.2.1 WRITE A SQL SELECT STATEMENT

- a) Write a SELECT statement to list the first and last names of all students.
- b) Write a SELECT statement to list all cities, states, and zip codes.
- c) Describe the result set of the following SQL statement:

```
SELECT *
FROM grade_type
```

2.2.2 USE DISTINCT IN A SQL STATEMENT

- a) Why are the result sets of each of the following SQL statements the same?

```
SELECT letter_grade
FROM grade_conversion
```

```
SELECT DISTINCT letter_grade
FROM grade_conversion
```

- b) Explain the result set of the following SQL statement:

```
SELECT DISTINCT cost
FROM course
```

 Lab 2.2: The Anatomy of a SELECT Statement **73**

- c) Explain what happens, and why, when you execute the following SQL statement:

```
SELECT DISTINCT course_no
FROM class
```

**LAB
2.2**

LAB 2.2 EXERCISE ANSWERS

2.2.1 ANSWERS

- a) Write a SELECT statement to list the first and last names of all students.

Answer: The SELECT list contains the two columns that provide the first and last names of students; the FROM clause lists the STUDENT table where these columns are found.

```
SELECT first_name, last_name
FROM student
```

FIRST_NAME	LAST_NAME
-----	-----
George	Eakheit
Leonard	Millstein
...	
Kathleen	Mastandora
Angela	Torres

268 rows selected.

You will also notice many rows are returned; you can examine each of the rows by scrolling up and down. There are many SET options in SQL*Plus that allow you to change the headings and the overall display of the data. As you work your way through this book, you will examine and learn about the most important SQL*Plus settings.

- b) Write a SELECT statement to list all cities, states, and zip codes.

Answer: The SELECT list contains the three columns that provide the city, state, and zip code; the FROM clause contains the ZIPCODE table where these columns are found.

```
SELECT city, state, zip
FROM zipcode
```

CITY	ST ZIP
-----	-----
Santurce	PR 00914

74 Lab 2.2: The Anatomy of a SELECT Statement

```

North Adams          MA 01247
...
New York             NY 10005
New York             NY 10035

```

227 rows selected.

LAB 2.2

- c) Describe the result set of the following SQL statement:

```

SELECT *
FROM grade_type

```

*Answer: All columns and rows of the GRADE_TYPE table are returned in the result set. If you use iSQL*Plus, your result will look similar to Figure 2.12. If you use SQL*Plus, your result may resemble the first listing of SQL output in Figure 2.13.*

FORMATTING YOUR RESULT: THE SQL*PLUS COLUMN AND FORMAT COMMANDS

If you are using SQL*Plus, not iSQL*Plus, you will notice that the result set is difficult to read when data “wraps” itself onto the next line. The result may look similar to the screen you see in Figure 2.13. This will often occur when your SELECT

The screenshot shows the Oracle iSQL*Plus web interface. At the top, the Oracle logo and 'iSQL*Plus' text are visible. On the right, there are icons for 'Logout', 'Preferences', and 'Help'. Below these is a 'Workspace' tab and 'History' link. The main area is titled 'Workspace' and contains a text input field with the SQL query: 'SELECT * FROM grade_type'. Below the input field are buttons for 'Execute', 'Load Script', 'Save Script', and 'Cancel'. The results are displayed in a table with the following data:

GRADE_	DESCRIPTION	CREATED_BY	CREATED_DATE	MODIFIED_BY	MODIFIED_DATE
FI	Final	MCAFFREY	31-DEC-98	MCAFFREY	31-DEC-98
HM	Homework	MCAFFREY	31-DEC-98	MCAFFREY	31-DEC-98
MT	Midterm	MCAFFREY	31-DEC-98	MCAFFREY	31-DEC-98
PA	Participation	MCAFFREY	31-DEC-98	MCAFFREY	31-DEC-98
PJ	Project	MCAFFREY	31-DEC-98	MCAFFREY	31-DEC-98
QZ	Quiz	MCAFFREY	31-DEC-98	MCAFFREY	31-DEC-98

Below the table, it says '6 rows selected.' At the bottom right of the interface, there is a 'Clear' button.

Figure 2.12 ■ SELECT statement against the GRADE_TYPE table issued in iSQL*Plus.

Lab 2.2: The Anatomy of a SELECT Statement 75

```

Oracle SQL*Plus
File Edit Search Options Help

SQL> SELECT *
  2   FROM grade_type
  3 /

GR DESCRIPTION                                CREATED_BY                                CREATED_D
-----
MODIFIED_BY                                MODIFIED_
-----
FI Final                                     MCAFFREY                                31-DEC-98
MCAFFREY
HN Homework                                 MCAFFREY                                31-DEC-98
MCAFFREY
HT Midterm                                  MCAFFREY                                31-DEC-98
MCAFFREY
PA Participation                             MCAFFREY                                31-DEC-98
MCAFFREY
PJ Project                                  MCAFFREY                                31-DEC-98
MCAFFREY
QZ Quiz                                     MCAFFREY                                31-DEC-98
MCAFFREY

6 rows selected.

SQL> COL description FORMAT A13
SQL> COL created_by FORMAT A8
SQL> COL modified_by FORMAT A8
SQL> /

GR DESCRIPTION    CREATED_  CREATED_D  MODIFIED  MODIFIED_
-----
FI Final          MCAFFREY 31-DEC-98 MCAFFREY 31-DEC-98
HN Homework      MCAFFREY 31-DEC-98 MCAFFREY 31-DEC-98
HT Midterm       MCAFFREY 31-DEC-98 MCAFFREY 31-DEC-98
PA Participation  MCAFFREY 31-DEC-98 MCAFFREY 31-DEC-98
PJ Project       MCAFFREY 31-DEC-98 MCAFFREY 31-DEC-98
QZ Quiz          MCAFFREY 31-DEC-98 MCAFFREY 31-DEC-98

6 rows selected.

SQL>

```

Figure 2.13 ■ SELECT issued in SQL*Plus for Windows.

statement contains multiple columns. To help you view the output more easily, SQL*Plus offers a number of formatting commands.

The SQL*Plus COLUMN command allows you to specify format attributes for specific columns. Because the SQL statement contains three alphanumeric columns, format each using these SQL*Plus commands:

```

COL description FORMAT A13
COL created_by FORMAT A8
COL modified_by FORMAT A8

```

When you re-execute the SQL statement, the result is more readable, as you see in the last result set shown in Figure 2.13.

The DESCRIPTION column is formatted to display a maximum of 13 characters; the CREATED_BY and MODIFIED_BY columns are formatted to display 8 characters. If the values in the columns do not fit into the space allotted, the data will wrap within the column. The column headings get truncated to the specified length.

76 Lab 2.2: The Anatomy of a SELECT Statement

The format for the column stays in place until you either respecify the format for the columns, specifically clear the format for the column, or exit SQL*Plus. To clear all the column formatting, execute the `CLEAR COLUMNS` command in SQL*Plus.

LAB 2.2

The two DATE datatype columns of this statement, `CREATED_DATE` and `MODIFIED_DATE`, are not formatted by the `COL` command. By default, Oracle displays all DATE datatype columns with a 9-character width. You will learn about formatting columns with the DATE datatype in Chapter 4, “Date and Conversion Functions.”

FORMATTING NUMBERS

If the column is of a NUMBER datatype column, you can change the format with a *format model* in the `COLUMN` command. For example, the 9 in the format model `999.99` represents the numeric digits, so the number 100 is displayed as 100.00. You can add dollar signs, leading zeros, angle brackets for negative numbers, and round values to format the display to your desire.

```
COL cost FORMAT $9,999.99
SELECT DISTINCT cost
FROM course
      COST
-----
      $1,095.00
      $1,195.00
      $1,595.00
```

4 rows selected.

If you did not allot sufficient room for the number to fit in the column, SQL*Plus will show a # symbol instead.

```
COL cost FORMAT 999.99
      COST
-----
      #####
      #####
      #####
```

4 rows selected.

For more SQL*Plus `COLUMN FORMAT` commands, see Appendix C, “SQL*Plus Command Reference.”

Lab 2.2: The Anatomy of a SELECT Statement 77



Throughout this book you notice that the output is displayed in SQL*Plus rather than iSQL*Plus format. The reason for this is simply that it takes up less space in the book.

2.2.2 ANSWERS

LAB 2.2

- a) Why are the result sets of each of the following SQL statements the same?

```
SELECT letter_grade
FROM grade_conversion
```

```
SELECT DISTINCT letter_grade
FROM grade_conversion
```

Answer: The result sets are the same because the data values in the LETTER_GRADE column in the GRADE_CONVERSION table are not repeated; the LETTER_GRADE column is the primary key of the table, so by definition its values are already distinct.

- b) Explain the result set of the following SQL statement:

```
SELECT DISTINCT cost
FROM course
```

Answer: The result set contains four rows of distinct costs in the COURSE table, including the NULL value.

```
SET FEEDBACK 1
```

```
SELECT DISTINCT cost
FROM course
COST
```

```
-----
1095
1195
1595
```

4 rows selected.

Note that if you changed the feedback SQL*Plus environment variable to 1, using the SQL*Plus command SET FEEDBACK 1, the result will include the “4 rows selected.” statement. There is one row in the COURSE table containing a null value in the COST column. Even though null is an unknown value, DISTINCT recognizes one or more null values in a column as one distinct value when returning a result set.

78 Lab 2.2: The Anatomy of a SELECT Statement

- c) Explain what happens, and why, when you execute the following SQL statement:

```
SELECT DISTINCT course_no
FROM class
```

Answer: Oracle returns an error because a table named CLASS does not exist.

```
FROM class
*
ERROR at line 2:
ORA-00942: table or view does not exist
```

The asterisk in the error message indicates the error in the query. SQL is an exacting language. As you learn to write SQL, you will inevitably make mistakes. It is important to pay attention to the error messages returned to you from the database to learn from and correct your mistakes. This Oracle error message tells you that you referenced a table or a view does not exist in this database schema. (Views are discussed in Chapter 12, “Views, Indexes, and Sequences.”) Correct your SQL statement and execute it again.

**LAB
2.2****LAB 2.2 SELF-REVIEW QUESTIONS**

In order to test your progress, you should be able to answer the following questions.

- 1) The column names listed in the SELECT list must be separated by commas.
 - a) True
 - b) False
- 2) A SELECT list may contain all the columns in a table.
 - a) True
 - b) False
- 3) The asterisk may be used as a wildcard in the FROM clause.
 - a) True
 - b) False
- 4) The following statement contains an error:

```
SELECT courseno
FROM course
```

- a) True
- b) False

Answers appear in Appendix A, Section 2.2.

LAB 2.3

EDITING A SQL STATEMENT

LAB OBJECTIVES

LAB 2.3

After this lab, you will be able to:

- ✓ Edit a SQL Statement Using SQL*Plus Commands
- ✓ Edit a SQL Statement Using an Editor
- ✓ Save, Retrieve, and Run a SQL Statement in *iSQL*Plus*

THE LINE EDITOR

In *iSQL*Plus* you can easily edit your statement just as any text. Sometimes you may not have access to *iSQL*Plus*, therefore you must learn how to write and edit a statement using the SQL*Plus line editor.

When using SQL*Plus, you may have noticed that typing the same SQL statement over and over again to make a small change quickly becomes very tedious. You can use SQL*Plus's line editor to change your statement, indicating which line to change, then use a command to execute the change.

At the SQL prompt, type and execute the following statement to retrieve a list of course numbers:

```
SELECT course_no
FROM course
```

SQL*Plus stores the last SQL command you typed in what is referred to as the *SQL buffer*. You can re-execute a statement by just pressing the */*, which reruns the command. The statement stays in the buffer until you enter another SQL command. Use the SQL*Plus LIST command, or simply the letter L, to list the contents of the buffer. The semicolon or the slash, both of which execute the statement, are not stored in the buffer. The asterisk next to the number 2 indicates this is the current line in the buffer.

80 Lab 2.3: Editing a SQL Statement

```
SQL>LIST
 1 SELECT course_no
 2* FROM course
```

For example, if you want to retrieve a list of descriptions instead, simply change the column `course_no` to `description` using the line editor. To make a change, indicate to the line editor which line to make current. To change it to the first line, type the number 1 at the SQL prompt:

```
SQL> 1
 1* SELECT course_no
```

Just the first line of the two-line statement is displayed, and the asterisk indicates this is now the current line in the buffer. You can make a change to that line with the `CHANGE` command:

```
SQL>CHANGE/course_no/description
```

The newly changed line is presented back to you:

```
 1* SELECT description
```

The `CHANGE` command is followed by a forward slash, followed by the text you want to change, and separated from the new text with another forward slash. The abbreviated command for the `CHANGE` command is the letter `C`. You are now ready to execute your statement to produce the new result set. Because you are not typing the statement for the first time, you cannot use the semicolon. Type a forward slash to execute the statement instead. The forward slash will always execute the current SQL statement in the buffer. Remember that certain commands you have learned so far, such as the `LIST` command, are not SQL, but SQL*Plus commands. Only SQL statements are saved in the buffer, never SQL*Plus commands.

USING AN EDITOR IN SQL*PLUS FOR WINDOWS

Although handy, using SQL*Plus's line editor capabilities can still be tedious, especially as your SQL statements grow in size and complexity. You may also want to save some statements for later use. This is where a *text editor* becomes useful. A text editor is a software program with no ability to format the text, such as with boldface or italics. Notepad, a text editor that comes with the Microsoft Windows operating systems, is one example of a text editor and is referenced in this book. Any other text editor will work just as well. For more about setting the default editor in SQL*Plus, see Appendix C, "SQL*Plus Command Reference."

To use a text editor in SQL*Plus for Windows or a SQL*Plus version with the command line interface, simply execute the `EDIT` or `ED` command. This command will *invoke*, or open, the default editor currently set in SQL*Plus. When you use

Lab 2.3: Editing a SQL Statement 81

the EDIT command at the SQL prompt, SQL*Plus will stay open in the background and your text editor will be in the foreground, automatically displaying the SQL statement in the buffer. The file already has a name, which can also be set as a default in SQL*Plus. For quick editing of statements, simply make your changes here, save the file, and exit Notepad, which brings you back to SQL*Plus. If you wish to save the file for future reference, while still in Notepad select Save As to save the file with a different name and any extension you wish. It is common to save SQL files with a .sql extension.

If your editor puts a .txt after the file name (effectively creating a myfile.sql.txt file), change the Save As type to *All Files* instead of *Text documents (*.txt)*. Another way to ensure the file contains a .sql extension is to enclose the entire file name in quotes, (e.g., "myfile.sql" or if you want to include the path "c:\examples\myfile.sql"). Figure 2.14 displays the Save As dialog in SQL*Plus.

LAB 2.3



*Notice that when you invoke an editor, the SQL statement ends with a forward slash on a separate line at the end. SQL*Plus adds this character to the file so the file can be executed in SQL*Plus. When*

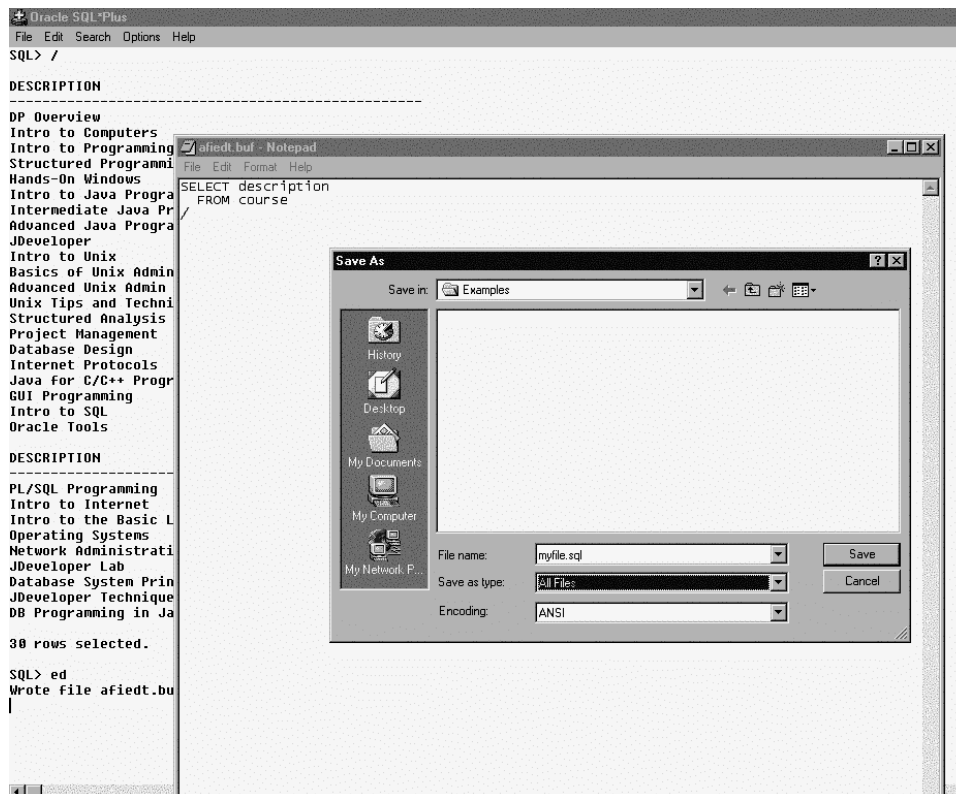


Figure 2.14 ■ Use of the Notepad text editor in SQL*Plus for Windows.

82 *Lab 2.3: Editing a SQL Statement*

*you invoke the editor from SQL*Plus, you can't go back to the SQL*Plus screen until you close the editor.*

Type the following statement:

```
SELECT *
FROM course
```

Now edit the file in Notepad and select Save As to save a second file with the name myfile2.sql. Exit Notepad and type and execute a new, different SQL statement:

```
SELECT state
FROM zipcode
```

This statement is now in the buffer; however, you can execute a different SQL statement, such as the one you saved in myfile2.sql, with the `START` or `@` command.

```
SQL>@myfile2
```

If the myfile2.sql file is stored in a directory other than the default directory, you need to specify the drive and directory name. You can also specify a valid URL such as `@http://script.repository/alice/myfile2.sql`.

```
SQL>@c:\examples\myfile2
```

The statement in the file runs, producing a result set. Because the file already contains a forward slash, the SQL statement is executed automatically. If you save myfile2 with an extension other than .sql, you must type the file name and extension. If you want to change myfile2 again, simply type the following. Notepad will open with myfile2.sql containing your SQL statement.

```
ED c:\examples\myfile2
```

CHANGING THE DEFAULT DIRECTORY OF SQL*PLUS FOR WINDOWS

Whenever you execute a script or save a file in SQL*Plus without specifying a directory, it is assumed to be in the default directory. Typically, this directory is named similar to `C:\oracle\product\10.1.0\Db_2\BIN` or `C:\oracle\ora10\BIN`. To change it to a different directory, such as the `c:\guest` directory, you need to create a shortcut. Modify the properties of the shortcut (see Figure 2.15) on the desktop to change the Start in field to the value `c:\guest` and then click OK. Whenever you invoke SQL*Plus through the shortcut, the `c:\guest` directory will be your default directory. If you are unsure how to create a shortcut in your Windows operating system, refer to the Windows documentation that came with your system. (Another way to change your default

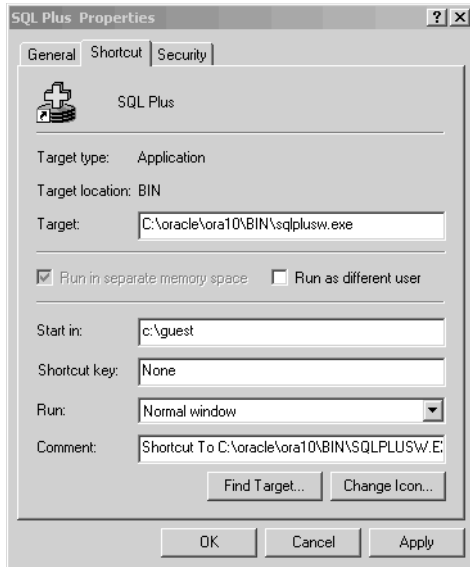


Figure 2.15 ■ Changing the default directory in SQL*Plus for Windows.

directory is by modifying your Windows registry. Only make these modifications if you are sufficiently knowledgeable about the Windows operating system. For more information, see Oracle's *SQL*Plus User's Guide and Reference Manual*.)

COPYING AND PASTING STATEMENTS IN SQL*PLUS FOR WINDOWS

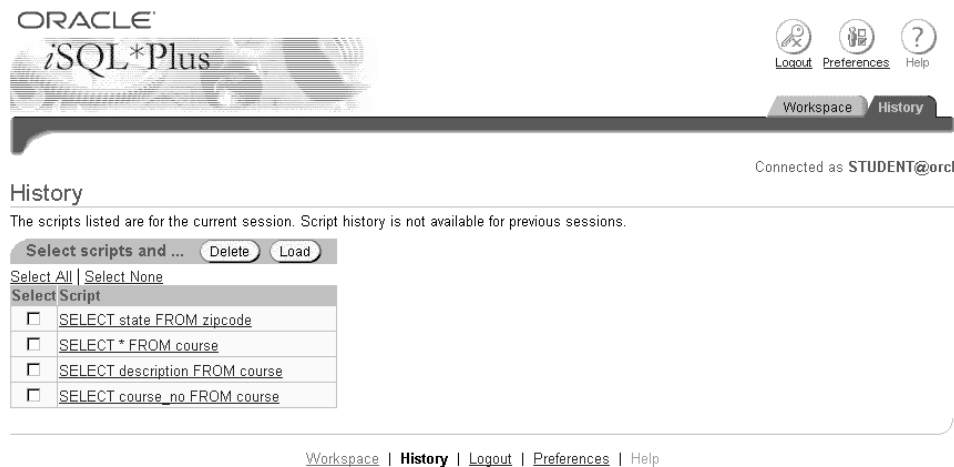
SQL*Plus for Windows allows you to copy and paste statements. You can open an editor such as Notepad in a separate window (without invoking it from SQL*Plus with the EDIT command) and enter your statements. Then select the text and copy the statement (CTRL + C or Copy from the Edit menu) and paste it into SQL*Plus using the Paste menu option or the CTRL + V command key.

EDITING IN iSQL*PLUS

Editing a SQL statement in *iSQL*Plus* is rather intuitive. You can enter the statements in the input area, also called the Workspace, and make changes using the delete and backspace keys or simply cut and paste. To save a statement to a text file, click on the Save Script button. You can reload the file later using the Load Script button. *iSQL*Plus* has a History tab that allows you to see the previously issued statements. Alternatively, you can also click the History link.

Figure 2.16 shows the last four statements that were issued. By default, at most the last ten statements or scripts are shown; you can increase this number when

84 Lab 2.3: Editing a SQL Statement



LAB 2.3

Figure 2.16 ■ History screen.

choosing Preferences, Interface Configuration, and then History Size. A history is only available for the duration of your current session. If you want to retain your statements after logout, you want to consider saving them to a file.

To run an individual statement, check the Select button and click the Load icon to bring the statement into the Workspace. You can load multiple statements at once, be sure to end each statement with a semicolon or a forward slash. You'll learn more about running multiple statements shortly.

PREFERENCES SCREEN

Either the Preferences icon on top or the Preference link on the bottom of the screen displays the Preferences screen. It allows you to customize your iSQL*Plus interface and execution environmental variables. Here you can change your password, modify the number of statements retained in the History screen, or change your display output location and page setup preferences. The menu choices on the left show Script Formatting, Script Execution, and Database Administration. These options allow you to change SQL*Plus environmental variables. You can leave them at their default setting. Most of these variables have equivalent SET commands that can be executed both in SQL*Plus and iSQL*Plus. You will learn more about these settings in Chapter 13, "The Data Dictionary and Advanced SQL*Plus Commands."

DIFFERENCES BETWEEN SQL*PLUS AND iSQL*PLUS

Throughout this book you will see both SQL*Plus and iSQL*Plus mentioned. For the most part the functionality between the two products is identical and does not impact on the result set, other than a different graphical output. If you are

Lab 2.3: Editing a SQL Statement 85

unclear if a certain SQL*Plus command performs identical in *iSQL*Plus*, refer to Appendix C, “SQL*Plus Command Reference.”

Overall, there are a small number of differences between the two products, particularly with respect to edits of SQL statements and the display of data. You will find these differences highlighted in Table 2.1.



*Unless specifically pointed out, all the mentioned SQL*Plus commands apply to both SQL*Plus and iSQL*Plus.*

LAB 2.3

Table 2.1 ■ Differences between SQL*Plus and iSQL*Plus

SQL*Plus	iSQL*Plus
Requires installation of SQL*Plus and Oracle Net software on individual machine.	No need to install special software, only browser is required.
Runs on individual workstation or on the server.	Runs from a browser, which is typically a workstation with access to the Web server where <i>iSQL*Plus</i> is running.
Editing via line editor or with your own editor.	Editing in the Workspace box.
SET commands control environmental variables that effect default formats and interface configuration settings among others.	Most of the SET commands can also be issued in <i>iSQL*Plus</i> . (See Appendix C, “SQL*Plus Command Reference” for differences). Alternatively, you can use the Preferences menu to modify the values.
Use the @ or START command to run scripts from a file or a URL.	For the @ or START command, only the URL format is supported.
Columns may not fit the whole width of your screen. Use various SQL*Plus formatting commands to make them display on one line.	The browser automatically handles the formatting of columns to fit the width of the screen.
To recall any previously issued statements use the SAVE command, write the statement to file, or scroll back and cut and paste.	To recall any previous statement, use the History tab, History link, or save the statement to file.

86 Lab 2.3: Editing a SQL Statement**LAB 2.3 EXERCISES**

*If you have access to only SQL*Plus but not iSQL*Plus or vice versa, just perform the exercises that are applicable for the specific environment. Exercises 2.3.1 and 2.3.2 use SQL*Plus only, not iSQL*Plus.*

**LAB
2.3****2.3.1 EDIT A SQL STATEMENT USING SQL*PLUS COMMANDS**

Type and execute the following SQL statement (use SQL*Plus, not iSQL*Plus):

```
SELECT employer
FROM student
```

- a)** Using SQL*Plus commands, change the column `employer` to `registration_date` and execute the statement again.
- b)** Using SQL*Plus commands, add a second column, `phone`, to the statement you changed. Display the `PHONE` column first, then the `REGISTRATION_DATE` column, in the result set.

2.3.2 EDIT A SQL STATEMENT USING AN EDITOR

Perform these exercises using SQL*Plus, not iSQL*Plus.

- a)** Invoke the editor and change the statement in your buffer to the following. Then save the file and execute it in SQL*Plus.

```
SELECT salutation, first_name, last_name, phone
FROM instructor
```

- b)** Edit the preceding statement, which is now in your buffer, save it as `inst.sql`, and use the `START` or `@` command to execute it in SQL*Plus.
- c)** Edit `inst.sql`, save it as `inst.x`, and use the `START` or `@` command to execute it in SQL*Plus.

2.3.3 SAVE, RETRIEVE, AND RUN A SQL STATEMENT IN iSQL*PLUS

- a)** Enter the following `SELECT` statement into the Workspace area and execute the statement. Then save the statement in a file called `state_zip.sql` and press the Clear button.

```
SELECT DISTINCT state
FROM zipcode
```

- b) Click the Load Script button and then the Browse... button and locate the state_zip.sql file you just saved. Then press the Load button to load it into the Workspace. Execute the statement.
- c) Explain the difference between the SQL language and SQL*Plus or iSQL*Plus.

LAB 2.3 EXERCISE ANSWERS

LAB 2.3

2.3.1 ANSWERS

Type and execute the following SQL statement (use SQL*Plus, not iSQL*Plus):

```
SELECT employer
FROM student
```

- a) Using SQL*Plus commands, change the column `employer` to `registration_date` and execute the statement again.

Answer: Select the first line in the buffer, then use the CHANGE command to change EMPLOYER to REGISTRATION_DATE.

Type 1 to select the first line in the buffer:

```
SQL> 1
      1* SELECT employer
```

Then use the CHANGE command:

```
SQL> c/employer/registration_date
      1* SELECT registration_date
```

Type L to list the changed statement:

```
SQL> L
      1 SELECT registration_date
      2* FROM student
```

If you care to run the query, you can do so with the forward slash "/", which then executes the statement currently in the buffer.

- b) Using SQL*Plus commands, add a second column, `phone`, to the statement you changed. Display the PHONE column first, then the REGISTRATION_DATE column, in the result set.

Answer: You must again select the first line in the buffer, then use the CHANGE command to add the PHONE column to the SELECT list.

88 *Lab 2.3: Editing a SQL Statement*

Type 1 to select the first line in the buffer:

```
SQL> 1
      1* SELECT registration_date
```

Then use the CHANGE command:

```
C/SELECT/SELECT phone,
```

Here, the CHANGE command will replace SELECT with SELECT phone, (including the comma), changing your statement to the following:

```
      1 SELECT phone, registration_date
      2* FROM student
```

The result set will display phone first, then the registration date:

```

PHONE                REGISTRAT
-----
201-555-5555          18-FEB-03
201-555-5555          22-FEB-03
...
718-555-5555          22-FEB-03
718-555-5555          28-JAN-03

268 rows selected.
```

The CHANGE command looks for the first occurrence, from left to right, of the text you wish to change. When it locates it, it replaces this occurrence with the new text you wish to change it to.

OTHER USEFUL LINE EDITOR COMMANDS

Besides the CHANGE and LIST commands, the SQL*Plus line editor has a number of other commands. For example, to add another column to the SQL statement you use the APPEND command. The statement currently in the buffer lists as follows:

```
SQL> L
      1 SELECT phone, registration_date
      2* FROM student
```

First choose the line to which you want to add at the end, then use the A command and add the text you want to append. In the following example the ", last_name" text was added to the statement.

 Lab 2.3: Editing a SQL Statement **89**

```
SQL> 1
1* SELECT phone, registration_date
SQL> A , last_name
1* SELECT phone, registration_date, last_name
```

Another useful command is the INPUT command; it adds a new line after the current line. To insert the text ", first_name" on the next line, use the INPUT or I command. SQL*Plus prompts you for a new line and you enter the text and press Enter. SQL*Plus prompts you once more for another new line and if you are finished adding, you press Enter again to indicate that you are done.

```
SQL> 1
1* SELECT phone, registration_date, last_name
SQL> I
2i      , first_name
3i
SQL> L
1 SELECT phone, registration_date, last_name
2      , first_name
3* FROM student
```

**LAB
2.3**

If you need to insert the line before line 1, enter a 0 (zero) followed by a space and text. Use the DEL command if you want to delete lines in the buffer. To delete line 2, you enter:

```
SQL> DEL 2
SQL> L
1 SELECT phone, registration_date, last_name
2* FROM student
```

You can also save the statement using the SQL*Plus SAVE command. In the next example, the SQL query is saved in the c:\guest directory under the file name myexample.sql; if you don't specify the extension, by default it will be .sql.

```
SQL> SAVE c:\guest\myexample
Created file c:\guest\myexample
```

You do not need to type a semicolon or forward slash, it will automatically be added. The statement can now be run either with the START or @ command. If you subsequently write other SQL statements and the statement is no longer in the SQL buffer, you can load it back into the buffer with the GET command. (The .sql extension is optional). You can then re-execute the statement with the forward slash.

```
SQL> GET c:\guest\myexample
1 SELECT phone, registration_date, last_name
2* FROM student
```

90 Lab 2.3: Editing a SQL Statement

2.3.2 ANSWERS

- a) Invoke the editor and change the statement in your buffer to the following. Then save the file and execute it in SQL*Plus.

```
SELECT salutation, first_name, last_name, phone
FROM instructor
```

*Answer: Use the EDIT command to edit the file and execute the changed statement in SQL*Plus with the forward slash.*

- b) Edit the preceding statement, which is now in your buffer, save it as inst.sql, and use the START or @ command to execute it in SQL*Plus.

*Answer: Use the EDIT command to edit the file and save it as inst.sql. Execute the changed statement in SQL*Plus with the START or @ command.*

```
SQL>@inst.sql
```

- c) Edit inst.sql, save it as inst.x, and use the START or @ command to execute it in SQL*Plus.

Answer: At the SQL prompt, type EDIT, edit the file in your editor, save the file as inst.x, exit the editor, type at the SQL> prompt the command @inst.x to execute the changed statement.

Because you saved the file with an extension other than .sql, you must explicitly reference both the file name and its extension. If you want to edit this file, you must type `EDIT inst.x` at the SQL prompt.

2.3.3 ANSWERS

- a) Enter the following SELECT statement into the Workspace area and execute the statement. Then save the statement in a file called state_zip.sql and press the Clear button.

```
SELECT DISTINCT state
FROM zipcode
```

Answer: When you execute this statement, it returns a list of the state abbreviations from the ZIPCODE table. When you click on the Save Script button, a message box informs you that the file is transferred from the Web browser to your individual computer. Click the Save button to save it on your computer (see Figure 2.17).

After you click the Save button you are prompted to enter the file name. You see a suggested file name, but change it to state_zip.sql instead and change the Save

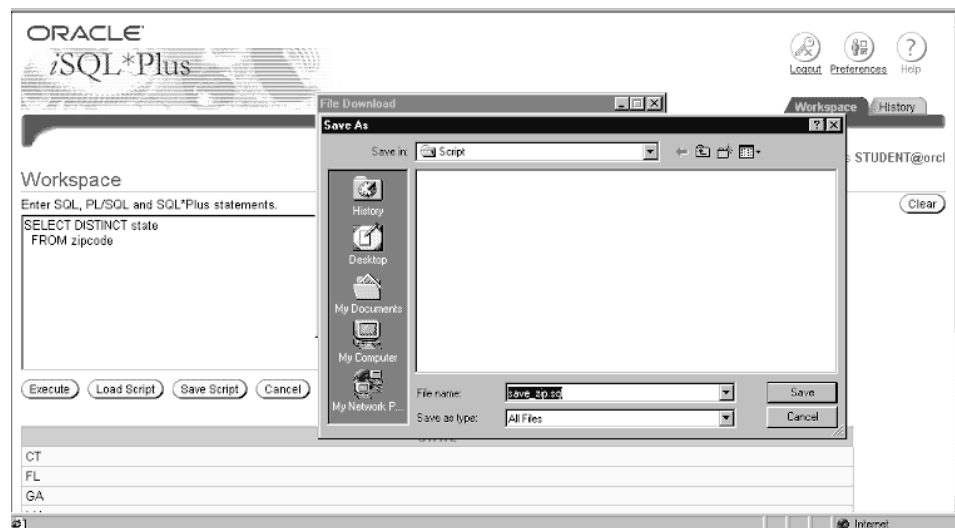


Figure 2.17 ■ Save a file in iSQL*Plus.

LAB 2.3

as type to “All Files”. The term *script* is just another word for command file containing one or multiple commands.

If you click the Clear button, the input area and output area are cleared, but note the SQL buffer is not cleared and it will still list the last statement if you were to enter the LIST command.

- b)** Click the Load Script button and then the Browse... button and locate the `state_zip.sql` file you just saved. Then press the Load button to load it into the Workspace. Execute the statement.

Answer: When you click the Browse... button, you will see a dialog box that displays the directory and file name similar to Figure 2.18. Then you need to press the Load button to transfer the file into the Workspace for execution. Afterwards, you can click the Execute button to run the statement. It is useful to save a file, if you want to retain the statement. iSQL*Plus retains a history of SQL statements you can access after you have cleared the screen, but it is no longer available after your session ends.

RUNNING MULTIPLE STATEMENTS IN iSQL*PLUS

You can run multiple SQL statements in iSQL*Plus. For example, if you want to run the following two statements, you either place them in a script file or simply type them into the input area. Just be sure to end every statement with either a semicolon or a forward slash at the beginning of a separate line. Note: you don't need a semicolon or forward slash for the last statement unless you run the statements inside a script file. In this case you must end each statement with either

92 Lab 2.3: Editing a SQL Statement

LAB 2.3

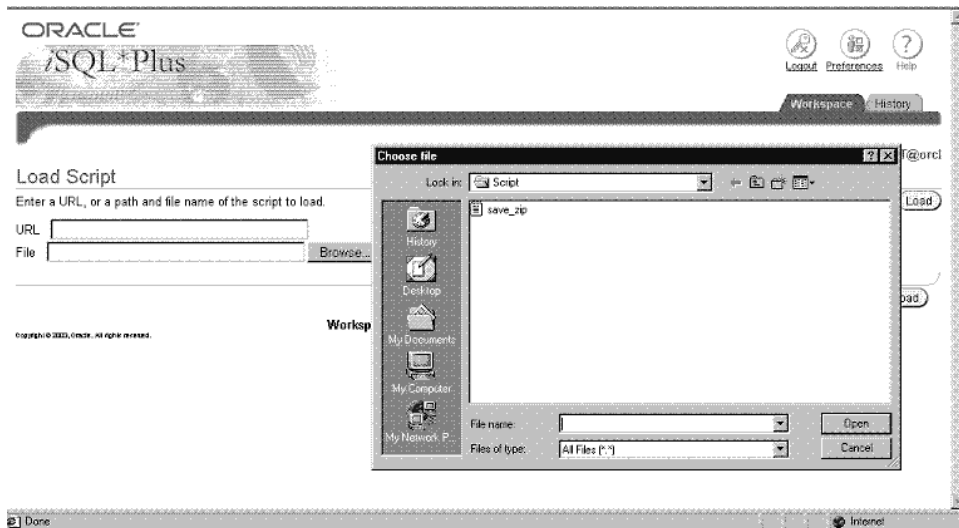


Figure 2.18 ■ The Choose file dialog in iSQL*Plus.

one. Therefore, it is a good habit to place either the semicolon or forward slash after each statement.

In Figure 2.19, you see two query results. One shows the distinct zip codes for all instructors, and the second result is a listing of first and last names for all students. The second statement does not quite fit on one screen, but you can scroll down to see the rest.

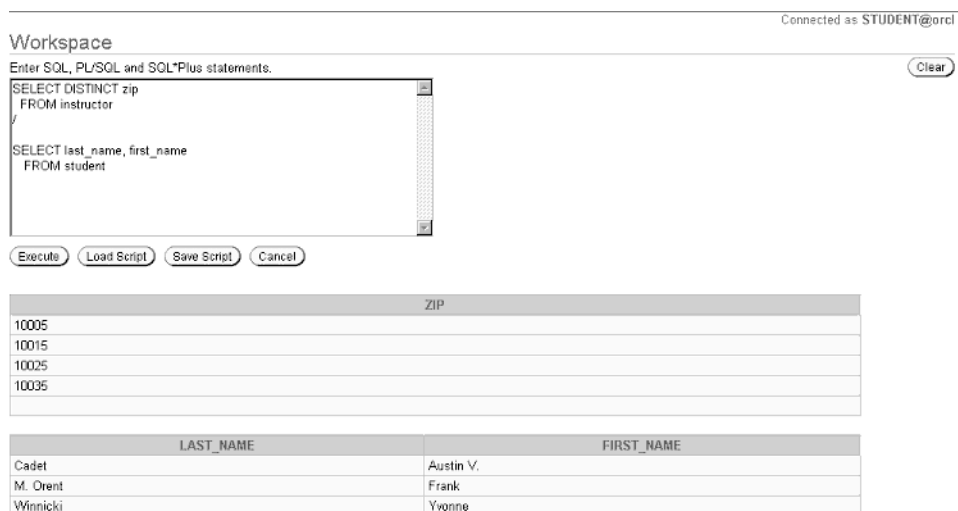


Figure 2.19 ■ Executing multiple SQL statements.

Lab 2.3: Editing a SQL Statement 93

Be careful, do not add both a semicolon and a forward slash to the same statement, otherwise it will be executed twice. For example, the next SQL statement will be executed twice.

```
SELECT DISTINCT zip
  FROM zipcode;
/
```

COMMENTS IN SQL SCRIPTS

Placing comments or remarks into a SQL script is very useful when you revisit the script later. It helps document the purpose, thoughts, and ideas or simply lists the author and creation date. You must identify the comment; otherwise you will receive an error when running the command. You can distinguish between two different types of comments: a single-line comment denoted with double dashes or a multiline comment spawning multiple lines, which starts with an opening comment like this, `/*`, and ends with a closing comment, which looks like this, `*/`.

Following is an example of a script file that includes comments, but comments can also be embedded within the SQL statement itself.

```
/* Multi-line comment
Homework #2
By: Kirsten Soehner
Date created: 4/30/2002
*/

-- Answer #1: This is a single-line comment!
SELECT DISTINCT state
  FROM zipcode;

-- Answer #2:
COL cost FORMAT $9,999.99
SELECT DISTINCT cost
  FROM course;

-- Answer #3:
SELECT instructor_id, -- Comment within a SQL statement!
       zip
  FROM instructor;
```

Note that SQL*Plus also has a `REMARK` command abbreviated as `REM` that allows single-line comments. This command is not recognized as a comment when your SQL statement is executed in an environment other than SQL*Plus or *iSQL*Plus*; it can also not be embedded in a SQL statement. Use the single-line and double-line comments mentioned previously instead!

94 Lab 2.3: Editing a SQL Statement

- c) Explain the difference between the SQL language and SQL*Plus or iSQL*Plus.

*Answer: SQL is a language that allows you to retrieve, manipulate, define, and control access to the database. SQL*Plus and iSQL*Plus are environments in which to execute the SQL statements and display the results.*

iSQL*Plus is the Web-based version of SQL*Plus and both programs are Oracle proprietary products. You can use other software programs to execute your SQL statements against an Oracle database. If you want to format your results in special ways use the SQL*Plus commands such as COLUMN FORMAT. If you don't execute the commands in iSQL*Plus or SQL*Plus these formatting commands are not available and you will need to use specific SQL functions to achieve somewhat similar results. Some of these SQL functions are discussed in the next chapter.

LAB 2.3

LAB 2.3 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

- 1) You can save a SQL statement to the SQL buffer for it to be referenced later.
 - a) True
 - b) False
- 2) After typing a SQL statement, you can execute it with either the semicolon or the forward slash.
 - a) True
 - b) False
- 3) You cannot save a .sql file to the A: drive.
 - a) True
 - b) False
- 4) The SQL*Plus START command can execute what is in the SQL buffer.
 - a) True
 - b) False

Answers appear in Appendix A, Section 2.3.

LAB 2.4

THE WHERE CLAUSE: COMPARISON AND LOGICAL OPERATORS

LAB 2.4

LAB OBJECTIVES

After this lab, you will be able to:

- ✓ Use Comparison and Logical Operators in a WHERE Clause
- ✓ Use NULL in a WHERE Clause

The *WHERE clause*, also called the *predicate*, provides the power to narrow down the scope of data retrieved. In fact, most SQL statements you write will contain a WHERE clause.

COMPARISON OPERATORS

Comparison operators compare expressions. An *expression* can be a column of any datatype, a *string* or *text literal* (sometimes referred to as a *text constant* or *character literal*), a number, or any combination of these. An expression can also be a *function* or *mathematical computation*, which you will learn about in Chapter 3, “Character, Number, and Miscellaneous Functions.” An expression always results in a value.

TESTING FOR EQUALITY AND INEQUALITY

Comparison operators compare one expression with another expression. One of the most commonly used comparison operators is the *equal* operator, denoted by the = symbol. For example, if you are asked to provide the first name, last name,

96 Lab 2.4: The WHERE Clause: Comparison and Logical Operators

and phone number of a teacher with the last name of Schorin, you write the following SQL statement:

```
SELECT first_name, last_name, phone
   FROM instructor
  WHERE last_name = 'Schorin'
```

```
FIRST_NAME LAST_NAME PHONE
-----
```

Nina	Schorin	2125551212
-------------	----------------	-------------------

```
1 row selected.
```

Here, the column LAST_NAME is the left side of the equation and the text literal 'Schorin' is the right side. Single quotes are used around the text literal 'Schorin'. This statement will only retrieve rows from the INSTRUCTOR table that satisfy this condition in the WHERE clause. In this case, only one row is retrieved.

LAB 2.4

When you describe the INSTRUCTOR table, you see the datatype of the LAST_NAME column is VARCHAR2. This means the data contained in this column is alphanumeric. When two values are compared to each other, they must be of the same datatype; otherwise, Oracle returns an error. You will learn more about converting from one datatype to another in Chapter 4, "Date and Conversion Functions."

```
SQL> DESCR instructor
```

Name	Null?	Type
-----		-----
INSTRUCTOR_ID	NOT NULL	NUMBER (8)
SALUTATION		VARCHAR2 (5)
FIRST_NAME		VARCHAR2 (25)
LAST_NAME		VARCHAR2 (25)
STREET_ADDRESS		VARCHAR2 (50)
ZIP		VARCHAR2 (5)
PHONE		VARCHAR2 (15)
CREATED_BY	NOT NULL	VARCHAR2 (30)
CREATED_DATE	NOT NULL	DATE
MODIFIED_BY	NOT NULL	VARCHAR2 (30)
MODIFIED_DATE	NOT NULL	DATE

SQL is case insensitive when it comes to column names, table names, and keywords such as SELECT. (There are some exceptions with regard to column names and table names. For more information see Chapter 11, "Create, Alter, and Drop Tables.") When you compare a text literal to a database column, the case of the data must match exactly. The syntax of the following statement is correct, but it does not yield any rows because the instructor's last name is obviously not in the correct case.

Lab 2.4: The WHERE Clause: Comparison and Logical Operators 97

```
SELECT first_name, last_name, phone
       FROM instructor
       WHERE last_name = 'schorin'
```

no rows selected

Just as equality is useful, so is inequality.

```
SELECT first_name, last_name, phone
       FROM instructor
       WHERE last_name <> 'Schorin'
```

FIRST_NAME	LAST_NAME	PHONE
Fernand	Hanks	2125551212
Tom	Wojick	2125551212
...		
Marilyn	Frantzen	2125551212
Irene	Willig	2125551212

9 rows selected.

All rows except the one with the last name of 'Schorin', are retrieved. Inequality can also be expressed with the != notation.

THE GREATER THAN AND LESS THAN OPERATORS

The comparison operators >, <, >=, and <= can all be used to compare values in columns. In the following example, the >=, or *greater than or equal to*, operator is used to retrieve a list of course descriptions whose cost is greater than or equal to 1195:

```
SELECT description, cost
       FROM course
       WHERE cost >= 1195
```

DESCRIPTION	COST
DP Overview	1195
Intro to Computers	1195
...	
Database System Principles	1195
PL/SQL Programming	1195

26 rows selected.

The value 1195 is not enclosed in single quotes because it is a number literal.

98 *Lab 2.4: The WHERE Clause: Comparison and Logical Operators***THE BETWEEN COMPARISON OPERATOR**

The BETWEEN operator tests for a range of values:

```

SELECT description, cost
  FROM course
 WHERE cost BETWEEN 1000 AND 1100
DESCRIPTION                                COST
-----
Unix Tips and Techniques                    1095
Intro to Internet                          1095
Intro to the Basic Language                1095

3 rows selected.

```

**LAB
2.4**

BETWEEN is inclusive of both values defining the range; the result set includes courses that cost 1000 and 1100 and everything in between. The lower end of the range must be listed first.

If you use *iSQL*Plus* then your result may look similar to Figure 2.20. Note that the result is identical; the only difference is the formatting.

DESCRIPTION	COST
Unix Tips and Techniques	1095
Intro to Internet	1095
Intro to the Basic Language	1095

Figure 2.20 ■ *iSQL*Plus* result.

BETWEEN is most useful for number and date comparisons, but it can also be used for comparing text strings in alphabetical order. Date comparisons are discussed in Chapter 4, “Date and Conversion Functions.”

THE IN OPERATOR

The IN operator works with a *list of values*, separated by commas, contained within a set of parentheses. The following query looks for courses where the cost is either 1095 or 1595.

```

SELECT description, cost
  FROM course
 WHERE cost IN (1095, 1595)
DESCRIPTION                                COST
-----
Structured Programming Techniques          1595
Unix Tips and Techniques                    1095

```

Lab 2.4: The WHERE Clause: Comparison and Logical Operators 99

```
Intro to Internet                1095
Intro to the Basic Language      1095
```

4 rows selected.

THE LIKE OPERATOR

Another very useful comparison operator is LIKE, which performs pattern-matching using the percent (%) or underscore (_) characters as wildcards. The percent wildcard is used to denote multiple characters, while the underscore wildcard is used to denote a single character. The next query retrieves rows where the last name begins with the uppercase letter S and ends in anything else:

```
SELECT first_name, last_name, phone
   FROM instructor
  WHERE last_name LIKE 'S%'
FIRST_NAME LAST_NAME PHONE
-----
```

Nina	Schorin	2125551212
Todd	Smythe	2125551212

2 rows selected.

The % character may be placed at the beginning, end, or anywhere within the literal text, but always within the single quotes. This is also true of the underscore wildcard character, as in this statement:

```
SELECT first_name, last_name
   FROM instructor
  WHERE last_name LIKE '_o%'
FIRST_NAME LAST_NAME
-----
```

Tom	Wojick
Anita	Morris
Charles	Lowry

3 rows selected.

The WHERE clause returns only rows where the last name begins with any character, but the second letter must be a lowercase o. The rest of the last name is irrelevant.

NEGATING USING NOT

All the previously mentioned operators can be negated with the NOT comparison operator; for example, NOT BETWEEN, NOT IN, NOT LIKE.

100 Lab 2.4: The WHERE Clause: Comparison and Logical Operators

```
SELECT phone
FROM instructor
WHERE last_name NOT LIKE 'S%'
```

In the SQL statement the LAST_NAME column used in the WHERE clause doesn't appear in the SELECT list. There is no rule about columns in the WHERE clause having to exist in the SELECT list.



The LIKE operator works well for simple pattern matching. If your pattern is more complex, you may want to consider using Oracle's regular expression functionality discussed in Chapter 15, "Regular Expressions and Hierarchical Queries."

LAB 2.4

EVALUATING NULL VALUES

Recall that NULL means an unknown value. The IS NULL and IS NOT NULL operators evaluate whether a data value is NULL or not. The following SQL statement returns courses that do not have a prerequisite:

```
SELECT description, prerequisite
FROM course
WHERE prerequisite IS NULL
```

DESCRIPTION	PREREQUISITE
DP Overview	
Intro to Computers	
Java for C/C++ Programmers	
Operating Systems	

4 rows selected.



Null values represent the unknown; a null cannot be equal or unequal to any value or to another null. Therefore, always use the IS NULL or IS NOT NULL operator when testing for nulls. There are a few exceptions when nulls are treated differently and a null can be equal to another null. One such example is the use of DISTINCT (see Lab 2.2). You will learn about the exceptions in the treatment of nulls throughout this book.

OVERVIEW OF COMPARISON OPERATORS

The comparison operators you have learned about so far are sometimes referred to as predicates or search conditions. A predicate is an expression that results in either a true, false, or unknown value. Table 2.2 provides you with a list of the

Lab 2.4: The WHERE Clause: Comparison and Logical Operators 101

Table 2.2 ■ SQL Comparison Operators

Comparison Operator	Definition
=	Equal
!=, <>	Not equal
>, >=	Greater than, greater than or equal to
<, <=	Less than, less than or equal to
BETWEEN . . . AND . . .	Inclusive of two values
LIKE	Pattern matching with wildcard characters % and _
IN (. . .)	List of values
IS NULL	Test for null values

LAB 2.4

most common comparison operators. You will learn about additional operators such as EXISTS, ANY, SOME, ALL in Chapter 7, “Subqueries,” and the OVERLAPS operator in Chapter 4, “Date and Conversion Functions.” All these operators can be negated with the NOT logical operator.

LOGICAL OPERATORS

To harness the ultimate power of the WHERE clause, comparison operators can be combined with the help of the *logical operators* AND and OR. These logical operators are also referred to as *boolean operators*. They group expressions, all within the same WHERE clause of a single SQL statement.

For example, the following SQL query combines two comparison operators with the help of the AND boolean operator. The result shows rows where a course costs 1095 and the course description starts with the letter I:

```
SELECT description, cost
  FROM course
 WHERE cost = 1095
       AND description LIKE 'I%'
```

DESCRIPTION	COST
-----	----
Intro to Internet	1095
Intro to the Basic Language	1095

2 rows selected.

With just the = operator in the WHERE clause, the result set contains three rows. With the addition of the AND description LIKE 'I%', the result is further reduced to two rows.

102 Lab 2.4: The WHERE Clause: Comparison and Logical Operators**PRECEDENCE OF LOGICAL OPERATORS**

When AND and OR are used together in a WHERE clause, the AND operator always takes precedence over the OR operator, meaning any AND conditions are evaluated first. If there are multiple operators of the same precedence, the left operator is executed before the right. You can manipulate the precedence in the WHERE clause with the use of parentheses. In the following SQL statement, the AND and OR logical operators are combined:

```
SELECT description, cost, prerequisite
FROM course
WHERE cost = 1195
      AND prerequisite = 20
      OR prerequisite = 25
```

DESCRIPTION	COST	PREREQUISITE
Hands-On Windows	1195	20
Structured Analysis	1195	20
Project Management	1195	20
GUI Programming	1195	20
Intro to SQL	1195	20
Intro to the Basic Language	1095	25
Database System Principles	1195	25

7 rows selected.

The preceding SQL statement selects any record that has either a cost of 1195 and a prerequisite of 20, or just has a prerequisite of 25 no matter what the cost. The sixth row, Intro to the Basic Language, is selected because it satisfies the OR expression prerequisite = 25. The seventh row, Database System Principles, only satisfies one of the AND conditions, not both. However, the row is part of the result set because it satisfies the OR condition.

Here is the same SQL statement, but with parentheses to group the expressions in the WHERE clause:

```
SELECT description, cost, prerequisite
FROM course
WHERE cost = 1195
      AND (prerequisite = 20
          OR prerequisite = 25)
```

DESCRIPTION	COST	PREREQUISITE
Database System Principles	1195	25
Hands-On Windows	1195	20
Structured Analysis	1195	20
Project Management	1195	20

Lab 2.4: The WHERE Clause: Comparison and Logical Operators **103**

GUI Programming	1195	20
Intro to SQL	1195	20

6 rows selected.

The first expression selects only courses where the cost is equal to 1195. If the prerequisite is either 25 or 20, then the second condition is also true. Both expressions need to be true for the row to be displayed. These are the basic rules of logical operators. If two conditions are combined with the AND operator, both conditions must be true; if two conditions are connected by the OR operator, only one of the conditions needs to be true for the record to be selected.

The result set returns six rows instead of seven. The order in which items in the WHERE clause are evaluated is changed by the use of parentheses and results in a different output.



To ensure that your SQL statements are clearly understood, it is always best to use parentheses.

LAB 2.4

NULLS AND LOGICAL OPERATORS

SQL uses *tri-value logic*; this means a condition can evaluate to true, false, or unknown. (This is in contrast to boolean logic, where a condition must be either true or false.) A row gets returned when the condition evaluates to true. The following query returns rows from the COURSE table starting with the words `Intro to` as the description *and* a value equal or larger than 140 in the PREREQUISITE column.

```
SELECT description, prerequisite
FROM course
WHERE description LIKE 'Intro to%'
AND prerequisite >= 140
```

DESCRIPTION	PREREQUISITE
Intro to Programming	140
Intro to Unix	310

2 rows selected.

Rows with a null value in the PREREQUISITE column are not included because null is an unknown value. This null value in the column is not equal or greater to 140. Therefore, the row `Intro to Computers` does not satisfy *both* conditions and is excluded from the result set. Following is the list of course descriptions with null values in the PREREQUISITE column. It shows the row `Intro to Computers` with a null value in the PREREQUISITE column.

104 Lab 2.4: The WHERE Clause: Comparison and Logical Operators

```
SELECT description, prerequisite, cost
FROM course
WHERE prerequisite IS NULL
```

DESCRIPTION	PREREQUISITE	COST
Operating Systems		1195
Java for C/C++ Programmers		1195
DP Overview		1195
Intro to Computers		1195

4 rows selected.

The AND truth table in Table 2.3 illustrates the combination of two conditions with the AND operator. Only if *both* conditions are true is a row returned for output. In this example, with the prerequisite being null, the condition is unknown and therefore the row not included in the result. The comparison against a null value yields unknown unless you specifically test for it with the IS NULL or IS NOT operators.

**LAB
2.4****Table 2.3 ■ AND Truth Table**

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

For the OR condition, just *one* of the conditions needs to be true. Again, let's examine how nulls behave under this scenario using the same query, but this time with the OR operator. The Intro to Computers course is now listed because it satisfies the 'Intro to%' condition only. In addition, you will notice that rows such as DB Programming in Java do not start with the 'Intro to' as the description, but satisfy the second condition, which is a prerequisite of greater or equal to 140.

```
SELECT description, prerequisite
FROM course
WHERE description LIKE 'Intro to%'
OR prerequisite >= 140
```

DESCRIPTION	PREREQUISITE
DB Programming in Java	350
Database Design	420
Internet Protocols	310
Intro to Computers	

Lab 2.4: The WHERE Clause: Comparison and Logical Operators 105

Intro to Internet	10
Intro to Java Programming	80
Intro to Programming	140
Intro to SQL	20
Intro to Unix	310
Intro to the Basic Language	25
JDeveloper Techniques	350
Oracle Tools	220
Structured Programming Techniques	204

13 rows selected.

Table 2.4 shows the truth table for the OR operator; it highlights the fact that just one of the conditions need be true for the row to be returned in the result set. It is irrelevant if the second condition evaluates to false or unknown.

Table 2.4 ■ OR Truth Table

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

**LAB
2.4**

When you negate a condition with the NOT operator and the value you are comparing against is a null value, it also results in a null (see Table 2.5). The following query demonstrates that none of the null prerequisites are included in the result set.

```

SELECT description, prerequisite
  FROM course
 WHERE NOT prerequisite >= 140

```

DESCRIPTION	PREREQUISITE
-----	-----
Intro to Internet	10
GUI Programming	20
Intro to SQL	20
Hands-On Windows	20
Structured Analysis	20
Project Management	20
Intro to the Basic Language	25
Database System Principles	25
PL/SQL Programming	80
Intro to Java Programming	80
Intermediate Java Programming	120

106 Lab 2.4: The WHERE Clause: Comparison and Logical Operators

Advanced Java Programming	122
JDeveloper	122
JDeveloper Lab	125
Basics of Unix Admin	130
Network Administration	130
Advanced Unix Admin	132
Unix Tips and Techniques	134

18 rows selected.

Table 2.5 ■ NOT Truth Table

NOT	TRUE	FALSE	UNKNOWN
NOT	FALSE	TRUE	UNKNOWN

**LAB
2.4**

LAB 2.4 EXERCISES

2.4.1 USE COMPARISON AND LOGICAL OPERATORS IN A WHERE CLAUSE

- Write a SELECT statement to list the last names of students living either in zip code 10048, 11102, or 11209.
- Write a SELECT statement to list the first and last names of instructors with the letter “i” (either uppercase or lowercase) in their last name living in the zip code 10025.
- Does the following statement contain an error? Explain.

```
SELECT last_name
FROM instructor
WHERE created_date = modified_by
```

- What do you observe when you execute the following SQL statement?

```
SELECT course_no, cost
FROM course
WHERE cost BETWEEN 1500 AND 1000
```

- Execute the following query and determine how many rows the query returns.

```
SELECT last_name, student_id
FROM student
WHERE ROWNUM <= 10
```

Lab 2.4: The WHERE Clause: Comparison and Logical Operators **107**

2.4.2 USE NULL IN A WHERE CLAUSE

- a) Write a SELECT statement to list descriptions of courses with prerequisites and cost less than 1100.
- b) Write a SELECT statement to list the cost of courses without a prerequisite; do not repeat the cost.

LAB 2.4 EXERCISE ANSWERS

2.4.1 ANSWERS

- a) Write a SELECT statement to list the last names of students living either in zip code 10048, 11102, or 11209.

Answer: The SELECT statement selects a single column and uses the IN comparison operator in the WHERE clause.

```
SELECT last_name
   FROM student
  WHERE zip IN ('10048', '11102', '11209')
LAST_NAME
-----
Masser
Allende
Winnicki
Wilson
Williams
McLean
Lefkowitz
```

7 rows selected.

The statement can also be written using the equal operator (=), in combination with the logical operator OR, and yields the same result set:

```
SELECT last_name
   FROM student
  WHERE zip = '10048'
         OR zip = '11102'
         OR zip = '11209'
```

There will be times when a SELECT statement can be written more than one way. The preceding statements are logically equivalent.

**LAB
2.4**

108 Lab 2.4: The WHERE Clause: Comparison and Logical Operators

- b) Write a SELECT statement to list the first and last names of instructors with the letter “i” (either uppercase or lowercase) in their last name living in the zip code 10025.

Answer: The SELECT statement selects two columns and uses the LIKE, =, and the AND and OR logical operators, combined with parentheses, in the WHERE clause.

```
SELECT first_name, last_name
FROM instructor
WHERE (last_name LIKE '%i%' OR last_name LIKE '%I%')
AND zip = '10025'
```

FIRST_NAME	LAST_NAME
Tom	Wojick
Nina	Schorin

2 rows selected.

The LIKE operator must be used twice in this example because there is no way of knowing whether there is an upper or lowercase ‘i’ anywhere in the last name. You must test for both conditions, which cannot be done using a single LIKE operator. If one of the OR conditions is true, the expression is true.

If you need to search for the actual % symbol within a column value, you can use a SQL function or an escape character. You learn more about this in Chapter 3, “Character, Number, and Miscellaneous Functions.”

- c) Does the following statement contain an error? Explain.

```
SELECT last_name
FROM instructor
WHERE created_date = modified_by
```

Answer: Yes. The two columns in the WHERE clause are not the same datatype and the Oracle database returns an error when this statement is executed.

You will get an error similar to the following when you execute the statement.

```
SQL> SELECT last_name
2 FROM instructor
3 WHERE created_date = modified_by
4 /
WHERE created_date = modified_by
*
```

ERROR at line 3:

ORA-01858: a non-numeric character was found where a numeric was expected

Lab 2.4: The WHERE Clause: Comparison and Logical Operators 109

There are times when the datatypes of columns do not agree and you need to convert from one datatype to another. You will learn about these circumstances in Chapter 4, “Date and Conversion Functions.” (In this exercise example, data conversion is not fruitful because the data in these two columns is of a very different nature.)

d) What do you observe when you execute the following SQL statement?

```
SELECT course_no, cost
FROM course
WHERE cost BETWEEN 1500 AND 1000
no rows selected
```

Answer: The query returns no rows. Although there are courses that cost between 1000 and 1500, the BETWEEN clause requires the lower end of the range to be listed first. If the query is rewritten as follows, it returns rows.

```
SELECT course_no, cost
FROM course
WHERE cost BETWEEN 1000 AND 1500
```

LAB 2.4

BETWEEN AND TEXT LITERALS

As mentioned previously, BETWEEN is most often used for numbers and dates, which you will learn about in Chapter 4, “Date and Conversion Functions.” You can apply the BETWEEN functions to text columns as you see in the next example, which utilizes the BETWEEN operator with text literals W and Z. The query lists the student’s ID and the last name. Notice any students whose last name begins with the letter “Z” are not included, because the STUDENT table has no student with a last name of the single letter “Z”. If a student’s last name was spelled “waldo”, this student would not be included in the result, because the WHERE clause is only looking for last names that fall between the uppercase letters of W and Z.

```
SELECT student_id, last_name
FROM student
WHERE last_name BETWEEN 'W' AND 'Z'
STUDENT_ID LAST_NAME
-----
      142 Waldman
...
      241 Yourish

11 rows selected.
```

If you are looking for “waldo”, regardless of the case, use the OR operator to include both conditions.

110 Lab 2.4: The WHERE Clause: Comparison and Logical Operators

```
SELECT student_id, last_name
FROM student
WHERE last_name BETWEEN 'W' AND 'Z'
OR last_name BETWEEN 'w' AND 'z'
```

Here is another example of how you can use the BETWEEN and the >= and <= operators with text literals.

```
SELECT description
FROM grade_type
WHERE description BETWEEN 'Midterm' and 'Project'
```

This would be equivalent to:

```
SELECT description
FROM grade_type
WHERE description >= 'Midterm'
AND description <= 'Project'
```

```
DESCRIPTION
-----
Midterm
Participation
Project

3 rows selected.
```

LAB 2.4

e) Execute the following query and determine how many rows the query returns.

```
SELECT last_name, student_id
FROM student
WHERE ROWNUM <= 10
```

Answer: The query returns ten rows. The WHERE clause uses the pseudocolumn ROWNUM, which restricts the result to the first ten or less rows. A pseudocolumn is not a real column that exists on a table; you can select the column, but you cannot manipulate its value.

LAST_NAME	STUDENT_ID
Kocka	230
Jung	232
Mulroy	233
Brendler	234
Carcia	235
Tripp	236
Frost	237
Snow	238

Lab 2.4: The WHERE Clause: Comparison and Logical Operators III

Scrittorale 240
Yourish 241

10 rows selected.

The next statement shows the value of the ROWNUM pseudocolumn column in the SELECT list. The first row displays the ROWNUM value of 1, the second the ROWNUM value of 2, and so on. The ROWNUM pseudocolumn is useful if you want to limit the number of rows returned by a query. You will see additional examples of this and other pseudocolumns in subsequent chapters.

```
SELECT ROWNUM, last_name, student_id
FROM student
WHERE ROWNUM <= 10
```

ROWNUM	LAST_NAME	STUDENT_ID
1	Kocka	230
2	Jung	232
3	Mulroy	233
4	Brendler	234
...		
9	Scrittorale	240
10	Yourish	241

10 rows selected.

**LAB
2.4**

2.4.2 ANSWERS

- a) Write a SELECT statement to list descriptions of courses with prerequisites and cost less than 1100.

Answer: The SELECT statement uses the IS NOT NULL and less than (<) comparison operators in the WHERE clause.

```
SELECT description, cost, prerequisite
FROM course
WHERE prerequisite IS NOT NULL
AND cost < 1100
```

DESCRIPTION	COST	PREREQUISITE
Intro to Internet	1095	10
Intro to the Basic Language	1095	25
Unix Tips and Techniques	1095	134

3 rows selected.

Both conditions need to be true for the row to be returned. If the one of the conditions is not met, the row simply is not selected for output.

112 Lab 2.4: The WHERE Clause: Comparison and Logical Operators

- b) Write a SELECT statement to list the cost of courses without a prerequisite; do not repeat the cost.

Answer: The SELECT statement selects a single column in combination with DISTINCT, and uses the IS NULL comparison operator in the WHERE clause.

```
SELECT DISTINCT cost
  FROM course
 WHERE prerequisite IS NULL
      COST
-----
      1195
```

1 row selected.

LAB 2.4

LAB 2.4 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

- 1) Comparison operators always compare two values only.

- a) True
b) False

- 2) The BETWEEN operator uses a list of values.

- a) True
b) False

- 3) The following statement is incorrect:

```
SELECT first_name, last_name
  FROM student
 WHERE employer = NULL
```

- a) True
b) False

- 4) The following statement is incorrect:

```
SELECT description
  FROM course
 WHERE cost NOT LIKE (1095, 1195)
```

- a) True
b) False

Lab 2.4: The WHERE Clause: Comparison and Logical Operators 113

- 5) The following statement is incorrect:

```
SELECT city
FROM zipcode
WHERE state != 'NY'
```

- a) True
b) False

- 6) The following statement returns rows in the STUDENT table where the last name begins with the letters SM.

```
SELECT last_name, first_name
FROM student
WHERE last_name = 'SM%'
```

- a) True
b) False

Answers appear in Appendix A, Section 2.4.

**LAB
2.4**

114 Lab 2.5: The ORDER BY Clause**LAB 2.5****THE ORDER BY CLAUSE****LAB OBJECTIVES**

After this lab, you will be able to:

- ✓ Custom Sort Query Results

**LAB
2.5****USING THE ORDER BY CLAUSE**

Recall from Chapter 1, “SQL and Data,” that data is not stored in a table in any particular order. In all of the examples used thus far, the result sets display data in the order in which they happen to be returned from the database. However, you may want to view data in a certain order and the ORDER BY clause accomplishes this by ordering the data any way you wish.

For example, the following statement retrieves a list of course numbers and descriptions for courses without a prerequisite, in alphabetical order by their descriptions:

```
SELECT course_no, description
FROM course
WHERE prerequisite IS NULL
ORDER BY description
COURSE_NO DESCRIPTION
-----
      10 DP Overview
      20 Intro to Computers
     146 Java for C/C++ Programmers
     310 Operating Systems
```

4 rows selected.

By default, when the ORDER BY is used, the result set is sorted in *ascending* order; or you can be explicit by adding the abbreviation ASC after the column. If descending order is desired, the abbreviation DESC is used after the column in the ORDER BY clause:

 Lab 2.5: The ORDER BY Clause **115**

```

SELECT course_no, description
FROM course
WHERE prerequisite IS NULL
ORDER BY description DESC
COURSE_NO DESCRIPTION
-----
      310 Operating Systems
      146 Java for C/C++ Programmers
       20 Intro to Computers
       10 DP Overview
  
```

4 rows selected.

Instead of listing the name of the column to be ordered, you can list the sequence number of the column in the SELECT list. The next SQL statement returns the same result as the prior SQL statement, but uses a different ORDER BY clause. The number 2 indicates the second column of the SELECT list.

```

SELECT course_no, description
FROM course
WHERE prerequisite IS NULL
ORDER BY 2 DESC
  
```

A result set can be sorted by more than one column. The columns you wish to sort by need only be included in the ORDER BY clause, separated by commas. The ORDER BY clause is always the last clause in an SQL statement.

**LAB
2.5**

COLUMN ALIAS

A column alias can be used in the SELECT list to give a column or value an alias; it can also make the result much easier to read. In next example, different forms of a column alias are used to take the place of the column name in the result set. An alias may also contain one or more words or be spelled in exact case when enclosed in double quotes. The optional keyword AS can precede the alias name.

```

SELECT first_name first,
       first_name "First Name",
       first_name AS "First"
FROM student
WHERE zip = '10025'
FIRST                                First Name                                First
-----
Jerry                                Jerry                                Jerry
Nicole                              Nicole                              Nicole
Frank                                Frank                                Frank
  
```

3 rows selected.

116 Lab 2.5: The ORDER BY Clause

To format the column with the SQL*Plus COLUMN format, you must specify the alias in quotes as well. For example:

```
COL "First" FORMAT A13
```

You can also use the column alias to order by a specific column.

```
SELECT first_name first, first_name "First Name",
       first_name AS "First"
FROM student
WHERE zip = '10025'
ORDER BY "First Name"
```

FIRST	First Name	First
-----	-----	-----
Frank	Frank	Frank
Jerry	Jerry	Jerry
Nicole	Nicole	Nicole

3 rows selected.

LAB 2.5

DISTINCT AND ORDER BY

The ORDER BY clause often contains columns listed in the SELECT clause, but it is also possible to ORDER BY columns that are not selected. One exception is columns qualified using the DISTINCT keyword—if the SELECT list contains DISTINCT, the column(s) the keyword pertains to must also be listed in the ORDER BY clause.

The next example shows that the STUDENT_ID column is not a column listed in the DISTINCT SELECT list and therefore results in an Oracle error message.

```
SQL> SELECT DISTINCT first_name, last_name
2     FROM student
3     WHERE zip = '10025'
4     ORDER BY student_id
5     /
ORDER BY student_id
*
ERROR at line 4:
ORA-01791: not a SELECTed expression
```

NULL VALUES AND ORDER BY

The following statement orders the COST column by the default sort order. Note that the row with a COST column value of NULL is the last row in the sort order.

Lab 2.5: The ORDER BY Clause 117

```

SELECT DISTINCT cost
  FROM course
 ORDER BY cost
      COST
-----
      1095
      1195
      1595

```

4 rows selected.

You can change the default ordering of the nulls with the NULLS FIRST or NULLS LAST option in the ORDER BY clause as you see in the next statement. Here the requested order is to list the NULL value first followed by the other values in the default ascending sort order.

```

SELECT DISTINCT cost
  FROM course
 ORDER BY cost NULLS FIRST
      COST
-----
      1095
      1195
      1595

```

4 rows selected.

**LAB
2.5**

UNDERSTANDING ORACLE ERROR MESSAGES

As you begin to learn SQL, you will inevitably make mistakes when writing statements. Oracle returns an error number and error message to inform you of your mistake. Some error messages are easy to understand; others are not. While we cannot anticipate every possible error you may encounter, you will see that throughout the book I point out common mistakes. Here are some general guidelines when dealing with Oracle errors.

1. READ THE ORACLE ERROR MESSAGE CAREFULLY

Oracle will tell you on which line the error occurred.

```

SQL> SELECT salutation, first_name, las_name
      2     FROM student
      3     WHERE first_name = 'John'
      4     /

```

118 Lab 2.5: The ORDER BY Clause

```

SELECT salutation, first_name, las_name
                                *
ERROR at line 1:
ORA-00904: "LAS_NAME": invalid identifier

```

In this example the error is very easy to spot and the error message is self-explanatory. One of the column names is invalid, and Oracle points out the error by indicating the line number. The error is on line 1, and the asterisk indicates in what position within the line the error is found; it is the misspelled LAST_NAME column name.

2. RESOLVE ONE ERROR AT A TIME

Sometimes you may have multiple errors in a single SQL statement. The Oracle *parser*, which checks the syntax of all statements, starts checking from the end of the entire statement.

```

SQL> SELECT salutation, first_name, las_name
      2     FROM studen
      3     WHER first_name = 'John'
      4     /
      WHER first_name = 'John'
      *
ERROR at line 3:
ORA-00933: SQL command not properly ended

```

This type of error message may leave you clueless as to what could be wrong with this query. In fact, the statement contains three errors, one in each line. Because the parser works its way backwards, it complains about the first error on line 3. The position of the asterisk suggests that there is something wrong with the spelling of the FIRST_NAME column. But in fact, it is spelled correctly; otherwise, you would see the ORA-00904 invalid identifier error listed as in one of the previous examples complaining about the incorrect column name. The WHERE key word is missing the letter E; therefore, Oracle cannot interpret what you are attempting to do.

After you correct this error, you will see line 2 reported, exemplifying how the parser works its way backward.

```

SQL> SELECT salutation, first_name, las_name
      2     FROM studen
      3     WHERE first_name = 'John'
      4     /
      FROM studen
      *
ERROR at line 2:
ORA-00942: table or view does not exist

```

Here the table name is misspelled and Oracle indicates that such a table does not exist.

Lab 2.5: The ORDER BY Clause 119

The last error in the statement is found on line 1 and is the misspelled LAST_NAME column name. The parser will report this error as the last error. If you are unsure about the spelling of a column or table name, you can always use the DESCRIBE SQL*Plus command to list the column names and their respective data types, or you can refer to Appendix D, “Student Database Schema,” for a list of table and column names.

3. DOUBLE-CHECK THE SYNTAX OF YOUR STATEMENT

Simple typos, such as a stray period or comma, a missing space, or single quote, can cause very strange and seemingly unrelated error message that may have nothing to do with the problem. Therefore, carefully reread the statement or simply retype it. After looking at a statement for a long time, the error may not be apparent. Perhaps put it aside, take a break, and look at it with a fresh mind later, or ask someone for help in spotting the error.

4. LOOK UP THE ORACLE ERROR NUMBER

You can look up the Oracle error number in the *Oracle Database Error Messages Manual*. If the error starts with an ORA message type, it is typically a database-related error, whereas an error with an SP2 prefix indicates a SQL*Plus or iSQL*Plus specific error. Once you found the error in the manual, you will see the reason for the error and a recommended action on how to correct it. The recommended action may be general or very specific, once again depending on what type of error occurred.

Initially, the challenge may be finding the correct manual to look up the error message. Following are some suggestions on how to find this information. Besides looking at the online documentation that comes with your Oracle software and which is either found on your CDs or installed on your machine, you can also find the online manual on the Oracle Technology Network (OTN) Web site. Oracle offers a free subscription to the site, which includes a number of features such as access to the online manuals and discussion groups. The URL for OTN is <http://otn.oracle.com>; you must register first to become a member. Also refer to Appendix H, “Navigating through the Oracle Documentation” and Appendix G, “Resources.” These appendixes offer you tips on how to find the needed information.

In some operating systems such as Unix, Linux, and VMS, you can also use the Oracle program called oerr to look up the error message from the operating system prompt. This does not work in the Windows environment. For example, to look up the ORA-00939 error you type at the Unix operating system prompt (indicated with the \$ sign):

```
$ oerr ora 00939
00939, 00000, " too many arguments for function"
// *Cause: The function was referenced with too many arguments.
// *Action: Check the function syntax and specify only the
//           required number of arguments.
$
```

120 Lab 2.5: The ORDER BY Clause**LAB 2.5 EXERCISES****2.5.1 CUSTOM SORT QUERY RESULTS**

- a) Write a SELECT statement to list each city and zip code in New York or Connecticut. Sort the result in ascending order by zip code.
- b) Write a SELECT statement to list course descriptions and their prerequisite course numbers, sorted in ascending order by description. Do not list courses without a prerequisite.
- c) Show the salutation, first, and last name of students with the last name Grant. Order the result by salutation in descending order and the first name in ascending order.
- d) Execute the following query. What do you observe about the last row returned by the query?

```
SELECT student_id, last_name
FROM student
ORDER BY last_name
```

**LAB
2.5****LAB 2.5 EXERCISE ANSWERS****2.5.1 ANSWERS**

- a) Write a SELECT statement to list each city and zip code in New York or Connecticut. Sort the result in ascending order by zip code.

Answer: The SELECT statement selects two columns, uses the equal operator and OR logical operator to combine expressions in the WHERE clause, and uses ORDER BY with a single column to sort the results in ascending order.

```
SELECT city, zip
FROM zipcode
WHERE state = 'NY'
OR state = 'CT'
ORDER BY zip
```

CITY	ZIP
-----	-----
Ansonia	06401
Middlefield	06455
...	

 Lab 2.5: The ORDER BY Clause **121**

```
Hicksville          11802
Endicott            13760
```

142 rows selected.

Alternatively, the WHERE clause can be written as:

```
WHERE state IN ('NY', 'CT')
```

- b)** Write a SELECT statement to list course descriptions and their prerequisite course numbers, sorted in ascending order by description. Do not list courses without a prerequisite.

Answer: The following query shows the use of the IS NOT NULL comparison operator in the WHERE clause. The result is sorted by the DESCRIPTION column in ascending order.

```
SELECT description, prerequisite
   FROM course
  WHERE prerequisite IS NOT NULL
  ORDER BY description
```

DESCRIPTION	PREREQUISITE
-----	-----
Advanced Java Programming	122
Advanced Unix Admin	132
...	
Structured Programming Techniques	204
Unix Tips and Techniques	134

26 rows selected.

Alternatively, the ORDER BY clause can be written as:

```
ORDER BY 1
```

You can even use the column alias.

```
SELECT description "Descr", prerequisite
   FROM course
  WHERE prerequisite IS NOT NULL
  ORDER BY "Descr"
```

In most of the previous examples, you see the SELECT list is taking up one line only. By spreading it over several lines, it sometimes makes it easier to read and this is perfectly acceptable formatting. By putting elements in the SELECT list on separate lines, you control exactly when the next line begins and indent it for easy readability below the line above it. The following SELECT statement has multiple columns in the SELECT list.

122 Lab 2.5: The ORDER BY Clause

```

SELECT description, prerequisite,
       cost, modified_date
FROM course
WHERE prerequisite IS NOT NULL
ORDER BY description

```

DESCRIPTION	PREREQUISITE	COST	MODIFIED_
Advanced Java Programming		122 1195	05-APR-03
...			
Unix Tips and Techniques		134 1095	05-APR-03

26 rows selected.

- c) Show the salutation, first, and last name of students with the last name Grant. Order the result by salutation in descending order and the first name in ascending order.

Answer: The ORDER BY clause contains two columns, the SALUTATION and the FIRST_NAME. The salutation is sorted first in descending order. Within each salutation, the first name is sorted in ascending order.

```

SELECT salutation, first_name, last_name
FROM student
WHERE last_name = 'Grant'
ORDER BY salutation DESC, first_name ASC

```

SALUT	FIRST_NAME	LAST_NAME
Ms.	Eilene	Grant
Ms.	Verona	Grant
Mr.	Omaira	Grant
Mr.	Scott	Grant

4 rows selected.

Again, you can write the query also with this ORDER BY clause:

```
ORDER BY 1 DESC, 2 ASC
```

Or to use the default order for the second column, which is ASC and can be omitted:

```
ORDER BY 1 DESC, 2
```

If you give your column a column alias, you can also use the column alias in the ORDER BY clause.

```

SELECT salutation "Sal", first_name "First Name",
       last_name "Last Name"
FROM student

```

Lab 2.5: The ORDER BY Clause 123

```

WHERE last_name = 'Grant'
ORDER BY "Sal" DESC, "First Name" ASC
Sal  First Name      Last Name
-----
Ms.  Eilene          Grant
Ms.  Verona         Grant
Mr.  Omaira        Grant
Mr.  Scott         Grant

```

4 rows selected.

- d) Execute the following query. What do you observe about the last row returned by the query?

```

SELECT student_id, last_name
FROM student
ORDER BY last_name

```

Answer: The student with the STUDENT_ID of 206 has the last name entered in lowercase. When ordering the result set, the lowercase letters are listed after the uppercase letters.

```

STUDENT_ID LAST_NAME
-----
          119 Abdou
          399 Abdou
...
          184 Zuckerberg
          206 annunziato

```

268 rows selected.

**LAB
2.5**

LAB 2.5 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

- 1) The following is the correct order of all clauses in this SELECT statement:

```

SELECT ...
FROM ...
ORDER BY ...
WHERE ...

```

- a) _____ True
b) _____ False

124 *Lab 2.5: The ORDER BY Clause*

2) You must explicitly indicate whether an ORDER BY is ascending.

a) True

b) False

3) The following statement is correct:

```
SELECT *  
  FROM instructor  
  ORDER BY phone
```

a) True

b) False

4) The following statement is incorrect:

```
SELECT description "Description",  
       prerequisite AS prereqs,  
       course_no "Course#"  
  FROM course  
  ORDER BY 3, 2
```

a) True

b) False

5) You can order by a column you have not selected.

a) True

b) False

Answers appear in Appendix A, Section 2.5.

CHAPTER 2

TEST YOUR THINKING



The projects in this section are meant to have you utilize all of the skills that you have acquired throughout this chapter. The answers to these projects can be found at the companion Web site to this book, located at <http://authors.phptr.com/rischert3e>. Visit the Web site periodically to share and discuss your answers.

- 1) Invoke an editor from SQL*Plus; create a file called `first.sql` containing an SQL statement that retrieves data from the `COURSE` table for courses that cost 1195, and whose descriptions start with 'Intro', sorted by their prerequisites.
- 2) Create another file called `second.sql` that retrieves data from the `STUDENT` table for students whose last names begin with 'A', 'B', or 'C', and who work for 'Competrol Real Estate', sorted by their last names.
- 3) Create yet another file called `third.sql` that retrieves all the descriptions from the `GRADE_TYPE` table, for rows that were modified by the user `MCAFFREY`.
- 4) Execute each of the files, in the order they were created.

