CERTIFICATION OBJECTIVE 2.02

Execute a Basic SELECT Statement

The practical capabilities of the SELECT statement are realized in its execution. The key to executing any query language statement is a thorough understanding of its syntax and the rules governing its usage. This topic is discussed first. It is followed by a discussion of the execution of a basic query before expressions and operators, which exponentially increase the utility of data stored in relational tables, are introduced. Next, the concept of a null value is demystified, as its pitfalls are exposed. These topics will be covered in the following four sections:

- Syntax of the primitive SELECT statement
- Rules are meant to be followed
- SQL expressions and operators
- NULL is nothing

Syntax of the Primitive SELECT Statement

In its most primitive form, the SELECT statement supports the projection of columns and the creation of arithmetic, character, and date expressions. It also facilitates the elimination of duplicate values from the results set. The basic SELECT statement syntax is as follows:

SELECT * | {[DISTINCT] *column* | *expression* [*alias*],…}
FROM *table*;

The special keywords or reserved words of the SELECT statement syntax appear in uppercase. When using the commands, however, the case of the reserved words in your query statement does not matter. The reserved words cannot be used as column names or other database object names. SELECT, DISTINCT, and FROM are three keyword elements. A SELECT statement always comprises two or more clauses. The two mandatory clauses are the SELECT clause and the FROM clause. The pipe symbol | is used to denote OR. So you can read the first form of the above SELECT statement as:

SELECT *
FROM *table;*

In this format, the asterisk symbol (*) is used to denote all columns. SELECT * is a succinct way of asking the Oracle server to return all possible columns. It is used as a shorthand, time-saving symbol instead of typing in SELECT *column1, column2, column3, column4,…,columnX*, to select all the columns. The FROM clause specifies which table to query to fetch the columns requested in the SELECT clause.

You can issue the following SQL command to retrieve all the columns and all the rows from the REGIONS table in the HR schema:

```
SELECT * FROM REGIONS;
```

As shown in Figure 2-2, when this command is executed in SQL*Plus, it returns all the rows of data and all the columns belonging to this table. Use of the asterisk in a SELECT statement is sometimes referred to as a "blind" query because the exact columns to be fetched are not specified.

The second form of the basic SELECT statement has the same FROM clause as the first form, but the SELECT clause is different:

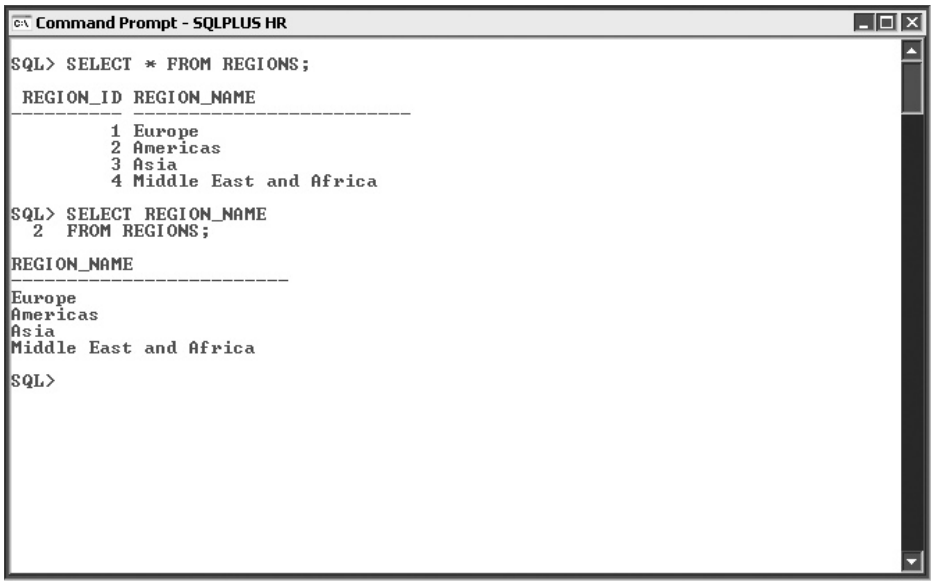
SELECT {[DISTINCT] *column* | *expression* [*alias*],…}
FROM *table;*

This SELECT clause can be simplified into two formats:

SELECT *column1 (possibly other columns or expressions)* [alias optional]
OR
SELECT DISTINCT *column1 (possibly other columns or expressions)* [alias optional]

FIGURE 2-2

Projecting all columns from the REGIONS table



```
SQL> SELECT * FROM REGIONS;

 REGION_ID REGION_NAME
---------- -------------------------
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa

SQL> SELECT REGION_NAME
  2  FROM REGIONS;

REGION_NAME
-------------------------
Europe
Americas
Asia
Middle East and Africa

SQL>
```

An *alias* is an alternative name for referencing a column or expression. Aliases are typically used for displaying output in a user-friendly manner. They also serve as shorthand when referring to columns or expressions to reduce typing. Aliases will be discussed in detail later in this chapter. By explicitly listing only the relevant columns in the SELECT clause you, in effect, *project* the exact subset of the results you wish to retrieve. The following statement will return just the REGION_NAME column subset of the REGIONS table as shown in Figure 2-2:

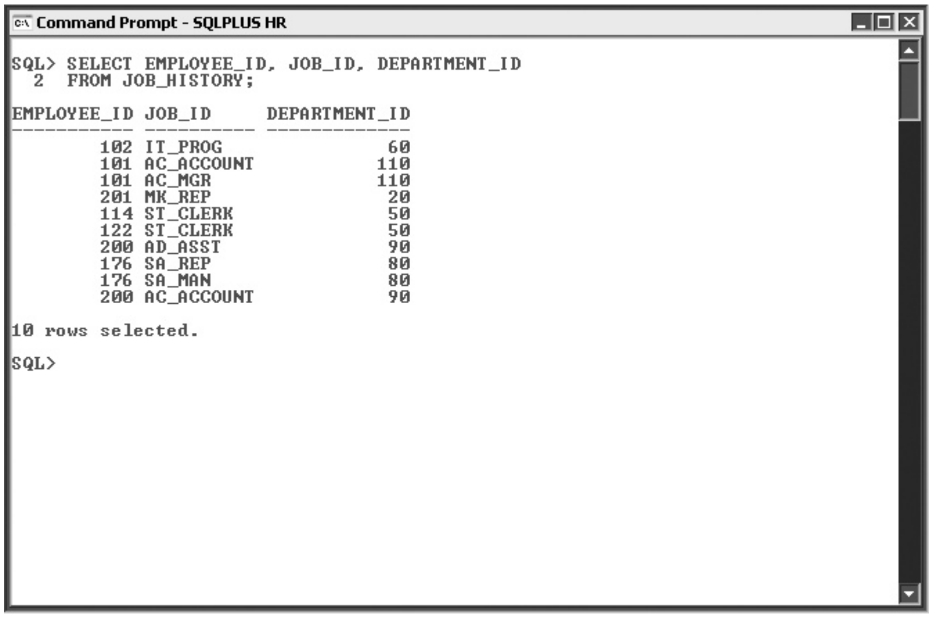
```
SELECT REGION_NAME
FROM REGIONS;
```

You may be asked to obtain all the job roles in the organization that employees have historically fulfilled. For this you can issue the command: SELECT * FROM JOB_HISTORY. However, in addition, the SELECT * construct returns the EMPLOYEE_ID, START_DATE, and END_DATE columns. The uncluttered results set containing only JOB_ID and DEPARTMENT_ID columns can be obtained with the following statement executed in SQL*Plus, as shown in Figure 2-3.

Using the DISTINCT keyword allows duplicate rows to be eliminated from the results set. In numerous situations a unique set of rows is required. It is important to note that the criterion employed by the Oracle server in determining whether a

FIGURE 2-3

Projecting specific columns from the JOB_HISTORY table



```
SQL> SELECT EMPLOYEE_ID, JOB_ID, DEPARTMENT_ID
  2  FROM JOB_HISTORY;

EMPLOYEE_ID JOB_ID     DEPARTMENT_ID
----------- ---------- -------------
        102 IT_PROG               60
        101 AC_ACCOUNT           110
        101 AC_MGR               110
        201 MK_REP                20
        114 ST_CLERK              50
        122 ST_CLERK              50
        200 AD_ASST               90
        176 SA_REP                80
        176 SA_MAN                80
        200 AC_ACCOUNT            90

10 rows selected.

SQL>
```

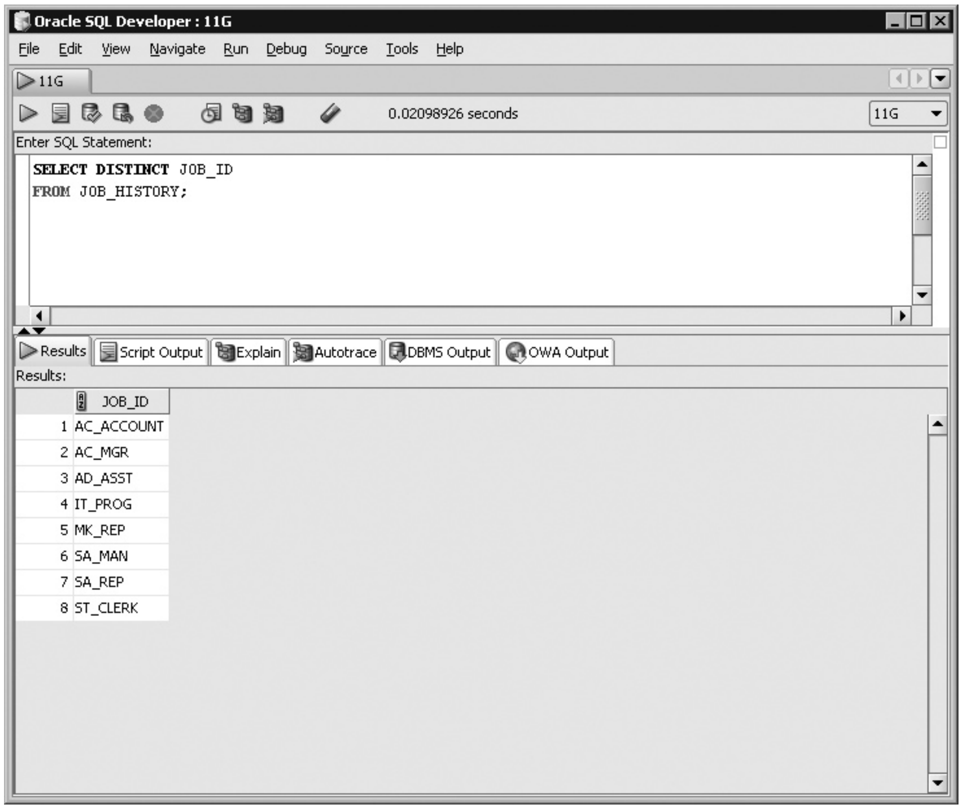
row is unique or distinct depends entirely on what is specified after the DISTINCT keyword in the SELECT clause. Selecting distinct JOB_ID values from the JOB_HISTORY table will return the eight distinct job types as shown in Figure 2-4.

Compare this output to Figure 2-3, where ten rows are returned. Can you see that there are two occurrences of the AC_ACCOUNT and ST_CLERK JOB_ID values? These are the two duplicate rows that have been eliminated by looking for distinct JOB_ID values. Selecting the distinct DEPARTMENT_ID column from the JOB_HISTORY table returns only six rows, as Figure 2-5 demonstrates. DEPARTMENT_ID values 50, 80, 90, and 110 each occur twice in the JOB_HISTORY table, and thus four rows have been eliminated by searching for distinct DEPARTMENT_ID values.

An important feature of the DISTINCT keyword is the elimination of duplicate values from *combinations* of columns. There are ten rows in the JOB_HISTORY table. Eight rows contain distinct JOB_ID values. Six rows contain distinct DEPARTMENT_ID values. Can you guess how many rows contain distinct combinations of JOB_ID and DEPARTMENT_ID values? As Figure 2-6 reveals,

FIGURE 2-4

Selecting unique JOB_IDs from the JOB_HISTORY table

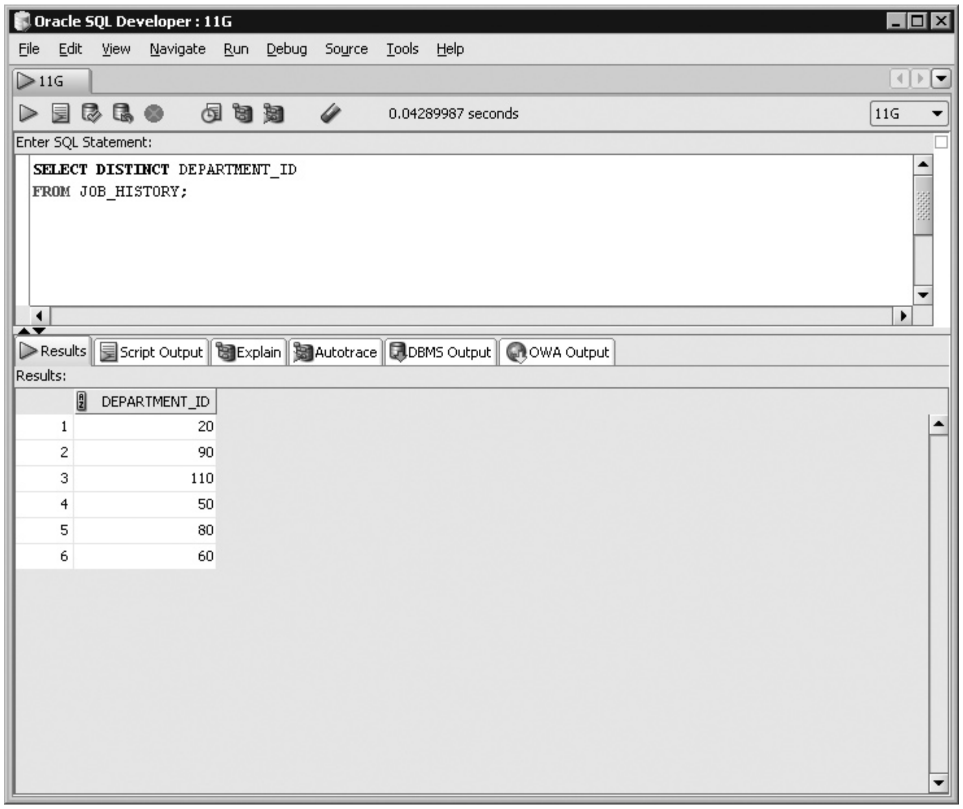

Oracle SQL Developer : 11G
File Edit View Navigate Run Debug Source Tools Help
11G
0.02098926 seconds
Enter SQL Statement:
Results Script Output Explain Autotrace DBMS Output OWA Output
Results:

```
SELECT DISTINCT JOB_ID
FROM JOB_HISTORY;
```

```
   JOB_ID
 1 AC_ACCOUNT
 2 AC_MGR
 3 AD_ASST
 4 IT_PROG
 5 MK_REP
 6 SA_MAN
 7 SA_REP
 8 ST_CLERK
```

FIGURE 2-5

Selecting unique DEPARTMENT_IDs from the JOB_HISTORY table


Oracle SQL Developer : 11G
File Edit View Navigate Run Debug Source Tools Help
11G
0.04289987 seconds
Enter SQL Statement:
Results Script Output Explain Autotrace DBMS Output OWA Output
Results:

```
SELECT DISTINCT DEPARTMENT_ID
FROM JOB_HISTORY;
```

```
   DEPARTMENT_ID
1             20
2             90
3            110
4             50
5             80
6             60
```

there are nine rows returned in the results set that contain distinct JOB_ID and DEPARTMENT_ID combinations, with one row from Figure 2-3 having being eliminated. This is, of course, the row that contains a JOB_ID value of ST_CLERK and a DEPARTMENT_ID value of 50.

on the job

The ability to project specific columns from a table is very useful. Coupled with the ability to remove duplicate values or combinations of values empowers you to assist with basic user reporting requirements. In many application databases, tables can sometimes store duplicate data. End user reporting frequently requires this data to be presented as a manageable set of unique records. This is something you now have the ability to do. Be careful, though, when using blind queries to select data from large tables. Executing a SELECT * FROM HUGE_TABLE; statement may cause performance issues if the table contains millions of rows of data.