

Limiting the columns retrieved by a `SELECT` statement is known as *projection* and was introduced in Chapter 2. Restricting the rows returned is known as *selection*. This chapter discusses the `WHERE` clause, which is an enhancement to the selection functionality of the `SELECT` statement. The `WHERE` clause specifies one or more conditions that the Oracle server evaluates to restrict the rows returned by the statement. A further language enhancement is introduced by the `ORDER BY` clause, which provides data sorting capabilities. Ampersand substitution introduces a way to reuse the same statement to execute different queries by substituting query elements at runtime. This area of runtime binding in SQL statements is thoroughly explored.

CERTIFICATION OBJECTIVE 3.01

Limit the Rows Retrieved by a Query

One of the cornerstone principles in relational theory is selection. Selection is actualized using the `WHERE` clause of the `SELECT` statement. Conditions that restrict the dataset returned take many forms and operate on columns as well as expressions. Only those rows in a table that conform to these conditions are returned. Conditions restrict rows using comparison operators in conjunction with columns and literal values. Boolean operators provide a mechanism to specify multiple conditions to restrict the rows returned. Boolean, conditional, concatenation, and arithmetic operators are discussed to establish their order of precedence when they are encountered in a `SELECT` statement. The following four areas are investigated:

- The `WHERE` clause
- Comparison operators
- Boolean operators
- Precedence rules

The `WHERE` clause

The `WHERE` clause extends the `SELECT` statement by providing the language to restrict rows returned based on one or more conditions. Querying a table with just the `SELECT` and `FROM` clauses results in every row of data stored in the table

being returned. Using the `DISTINCT` keyword, duplicate values are excluded, and the resultant rows are restricted to some degree. What if very specific information is required from a table, for example, only the data where a column contains a specific value? How would you retrieve the countries that belong to the Europe region from the `COUNTRIES` table? What about retrieving just those employees who work as sales representatives? These questions are answered using the `WHERE` clause to specify exactly which rows must be returned. The format of the SQL `SELECT` statement which includes the `WHERE` clause is:

```
SELECT * | [[DISTINCT] column | expression [alias],...]
FROM table
[WHERE condition(s)];
```

The `SELECT` and `FROM` clauses were examined in Chapter 2. The `WHERE` clause always follows the `FROM` clause. The square brackets indicate that the `WHERE` clause is optional. One or more conditions may be simultaneously applied to restrict the result set. A condition is specified by comparing two terms using a conditional operator. These terms may be column values, literals, or expressions. The *equality* operator is most commonly used to restrict result sets. Two examples of `WHERE` clauses are shown next:

```
select country_name
from countries
where region_id=3;

select last_name, first_name from employees
where job_id='SA_REP';
```

The first example projects the `COUNTRY_NAME` column from the `COUNTRIES` table. Instead of selecting every row, the `WHERE` clause restricts the rows returned to only those which contain a 3 in the `REGION_ID` column. The second example projects two columns, `LAST_NAME` and `FIRST_NAME` from the `EMPLOYEES` table. The rows returned are restricted to those which contain the value `SA_REP` in their `JOB_ID` columns.

Numeric-Based Conditions

Conditions must be formulated appropriately for different column data types. The conditions restricting rows based on numeric columns can be specified in several different ways. Consider the `SALARY` column in the `EMPLOYEES` table. This column has a data type of `NUMBER(8,2)`. Figure 3-1 shows two different ways in which the `SALARY` column has been restricted. The first and second examples

FIGURE 3-1

Two ways to
select numeric
values in a
WHERE clause

```

SQL> SELECT LAST_NAME, SALARY FROM EMPLOYEES
2  WHERE SALARY = 10000;

LAST_NAME          SALARY
-----
Baer                10000
Tucker              10000
King                10000
Bloom               10000

SQL> SELECT LAST_NAME, SALARY FROM EMPLOYEES
2  WHERE SALARY = '10000';

LAST_NAME          SALARY
-----
Baer                10000
Tucker              10000
King                10000
Bloom               10000

SQL> _

```

retrieve the LAST_NAME and SALARY values of the employees who earn \$ 10,000. Notice the difference in the WHERE clauses of the following queries. The first query specifies the number 10000, while the second encloses the number within single quotes like a character literal. Both formats are acceptable to Oracle since an implicit data type conversion is performed when necessary.

```

select last_name, salary from employees
where salary = 10000;

```

```

select last_name, salary from employees
where salary = '10000';

```

A numeric column can be compared to another numeric column in the same row to construct a WHERE clause condition, as the following query demonstrates:

```

select last_name, salary from employees
where salary = department_id;

```

The first example in Figure 3-2 shows how the WHERE clause is too restrictive and results in no rows being selected. This is because the range of SALARY values is 2100 to 999999.99, and the range of DEPARTMENT_ID values is 10 to 110. Since there is no overlap in the range of DEPARTMENT_ID and SALARY values, there are no rows that satisfy this condition and therefore nothing is returned. The example also illustrates how a WHERE clause condition compares one numeric column to another.

FIGURE 3-2

Using the
WHERE clause
with numeric
expressions

```

SQL> SELECT LAST_NAME, SALARY FROM EMPLOYEES
2  WHERE SALARY = DEPARTMENT_ID;

no rows selected

SQL> SELECT LAST_NAME, SALARY FROM EMPLOYEES
2  WHERE SALARY = DEPARTMENT_ID * 100;

LAST_NAME      SALARY
-----
Ernst           6000
Olsen           8000
Smith           8000

SQL>

```

The second example in Figure 3-2 demonstrates extending the WHERE clause condition to compare a numeric column, SALARY, to the numeric expression: DEPARTMENT_ID*100. For each row, the value in the SALARY column is compared to the product of the DEPARTMENT_ID value and 100. The WHERE clause also permits expressions on either side of the comparison operator. You could issue the following statement to yield identical results:

```

select last_name, salary from employees
where salary/10 = department_id*10;

```

As in regular algebra, the expression (SALARY = DEPARTMENT_ID * 100) is equivalent to (SALARY/10 = DEPARTMENT_ID * 10). The notable feature about this example is that the terms on either side of the comparison operator are expressions.

Character-Based Conditions

Conditions determining which rows are selected based on character data, are specified by enclosing character literals in the conditional clause, within single quotes. The JOB_ID column in the EMPLOYEES table has a data type of VARCHAR2(10). Suppose you wanted a report consisting of the LAST_NAME values of those employees currently employed as sales representatives. The JOB_ID value for a sales representative is SA_REP. The following statement produces such a report.

```

select last_name
from employees
where job_id='SA_REP';

```

If you tried specifying the character literal without the quotes, an Oracle error would be raised. Remember that character literal data is case sensitive, so the following WHERE clauses are not equivalent.

```
Clause 1: where job_id=SA_REP
Clause 2: where job_id='Sa_Rep'
Clause 3: where job_id='sa_rep'
```

Clause 1 generates an “ORA-00904: “SA_REP”: invalid identifier” error since the literal SA_REP is not wrapped in single quotes. Clause 2 and Clause 3 are syntactically correct but not equivalent. Further, neither of these clauses yields any data since there are no rows in the EMPLOYEES table which have JOB_ID column values that are either Sa_Rep or sa_rep, as shown in Figure 3-3.

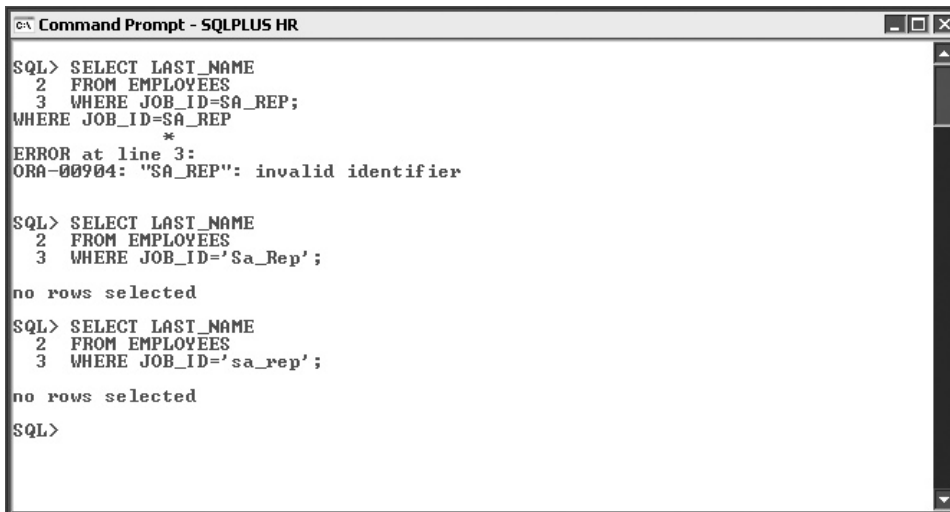
Character-based conditions are not limited to comparing column values with literals. They may also be specified using other character columns and expressions. The LAST_NAME and FIRST_NAME columns are both specified as VARCHAR2(25) data typed columns. Consider the query:

```
select employee_id, job_id
from employees
where last_name=first_name;
```

Both the LAST_NAME and FIRST_NAME columns appear on either side of the equality operator in the WHERE clause. No literal values are present; therefore no

FIGURE 3-3

Using the WHERE clause with character data



```

SQL> SELECT LAST_NAME
  2  FROM EMPLOYEES
  3  WHERE JOB_ID=SA_REP;
WHERE JOB_ID=SA_REP
*
ERROR at line 3:
ORA-00904: "SA_REP": invalid identifier

SQL> SELECT LAST_NAME
  2  FROM EMPLOYEES
  3  WHERE JOB_ID='Sa_Rep';

no rows selected

SQL> SELECT LAST_NAME
  2  FROM EMPLOYEES
  3  WHERE JOB_ID='sa_rep';

no rows selected

SQL>
```

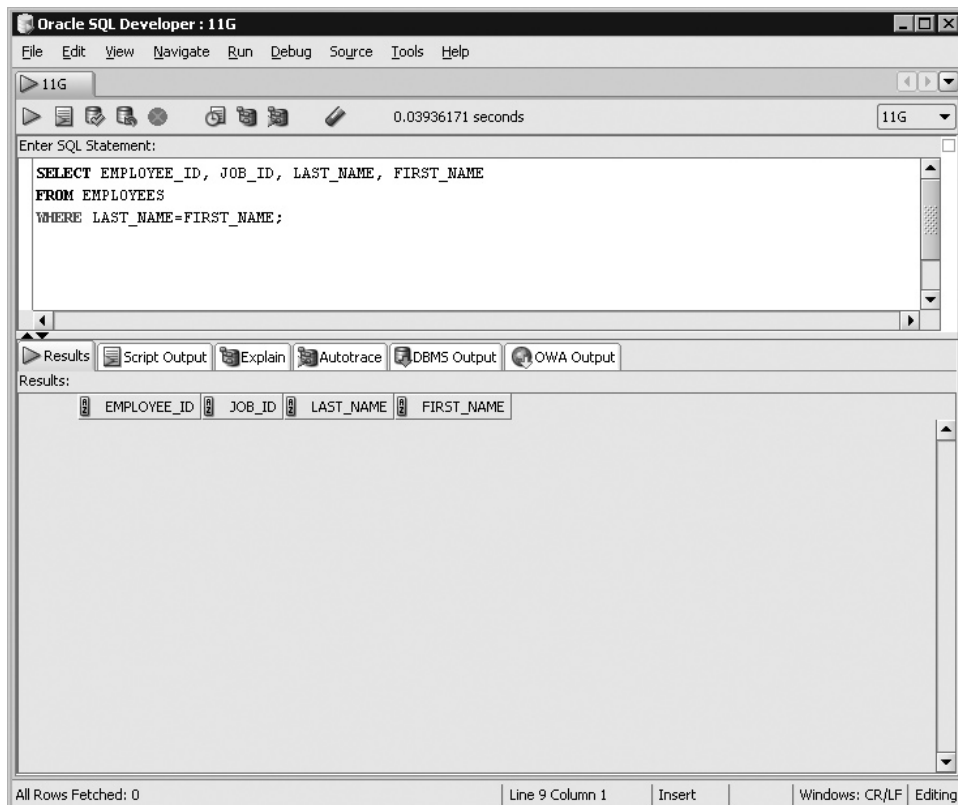
single quote characters are necessary to delimit them. This condition stipulates that only rows which contain the same data value (an exact case-sensitive match) in the LAST_NAME and FIRST_NAME columns will be returned. This condition is too restrictive and, as Figure 3-4 shows, no rows are returned.

Character-based expressions form either one or both parts of a condition separated by a conditional operator. These expressions can be formed by concatenating literal values with one or more character columns. The following four clauses demonstrate some of the options for character-based conditions:

```
Clause 1: where 'A '||last_name||first_name = 'A King'
Clause 2: where first_name||' '||last_name = last_name||' '||first_name
Clause 3: where 'SA_REP' || 'King' = job_id||last_name
Clause 4: where job_id||last_name ='SA_REP' || 'King'
```

FIGURE 3-4

Character
column-based
WHERE clause



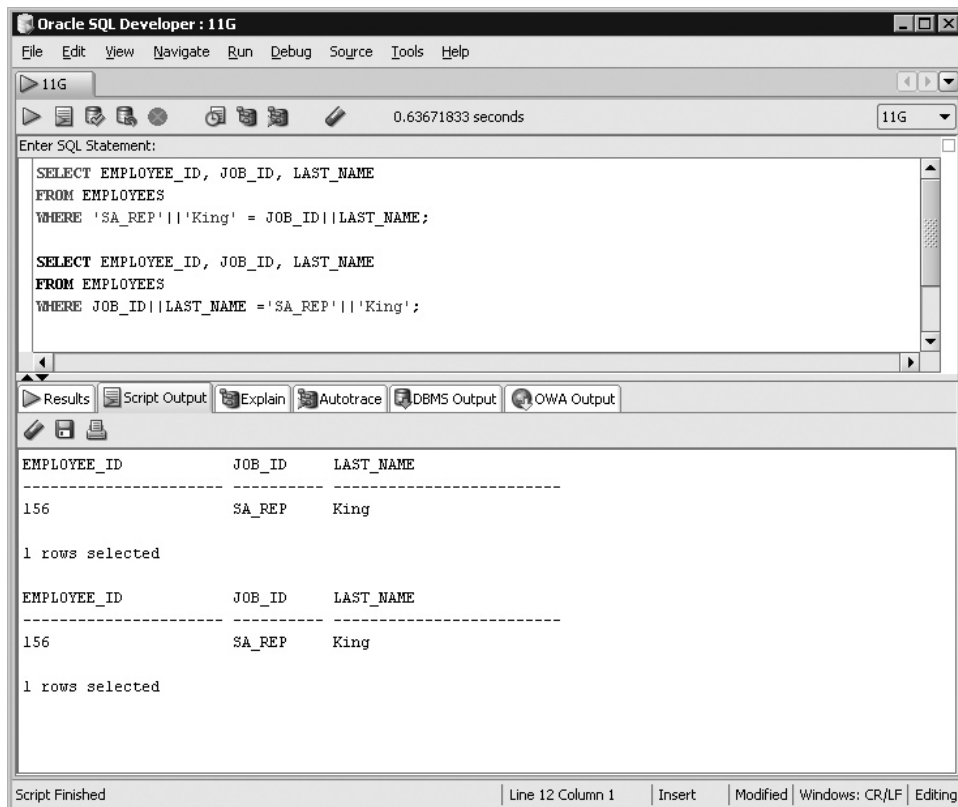
Clause 1 concatenates the string literal “A” to the LAST_NAME and FIRST_NAME columns. This expression is compared to the literal “A King,” and any row that fulfils this condition is returned. Clause 2 demonstrates that character expressions may be placed on both sides of the conditional operator. Clause 3 illustrates that literal expressions may also be placed on the left of the conditional operator. It is logically equivalent to clause 4, which has swapped the operands in clause 3 around. Both clauses 3 and 4 result in the same row of data being returned, as shown in Figure 3-5.

Date-Based Conditions

DATE columns are useful when storing date and time information. Date literals must be enclosed in single quotation marks just like character data; otherwise an error is raised. When used in conditional WHERE clauses, DATE columns are

FIGURE 3-5

Equivalence
of conditional
expressions



compared to other DATE columns or to date literals. The literals are automatically converted into DATE values based on the default date format, which is DD-MON-RR. If a literal occurs in an expression involving a DATE column, it is automatically converted into a date value using the default format mask. DD represents days, MON represents the first three letters of a month, and RR represents a Year 2000–compliant year (that is, if RR is between 50 and 99, then the Oracle server returns the previous century, else it returns the current century). The full four-digit year, YYYY, can also be specified. Consider the following four SQL statements:

```
Statement 1:
select employee_id from job_history
where start_date = end_date;
```

```
Statement 2:
select employee_id from job_history
where start_date = '01-JAN-2001';
```

```
Statement 3:
select employee_id from job_history
where start_date = '01-JAN-01';
```

```
Statement 4:
select employee_id from job_history
where start_date = '01-JAN-99';
```

The first statement tests equality between two DATE columns. Rows that contain the same values in their START_DATE and END_DATE columns will be returned. Note, however, that DATE values are only equal to each other if there is an exact match between all their components including day, month, year, hours, minutes, and seconds. Chapter 4 discusses the details of storing DATE values. Until then, don't worry about the hours, minutes, and seconds components.

In the WHERE clause of the second statement, the START_DATE column is compared to the character literal: '01-JAN-2001'. The entire four-digit year component (YYYY) has been specified. This is acceptable to the Oracle server, and all rows in the JOB_HISTORY table with START_DATE column values equal to the first of January 2001 will be returned.

The third statement is equivalent to the second since the literal '01-JAN-01' is converted to the date value 01-JAN-2001. This is due to the RR component being less than 50, so the current (twenty-first) century, 20, is prefixed to the year RR component to provide a century value. All rows in the JOB_HISTORY table with START_DATE column values = 01-JAN-2001 will be returned.

The century component for the literal '01-JAN-99' becomes the previous (twentieth) century, 19, yields a date value of 01-JAN-1999 for the fourth statement, since the RR component, 99, is greater than 50. Rows in the JOB_HISTORY table with START_DATE column values = 01-JAN-1999 will be returned.

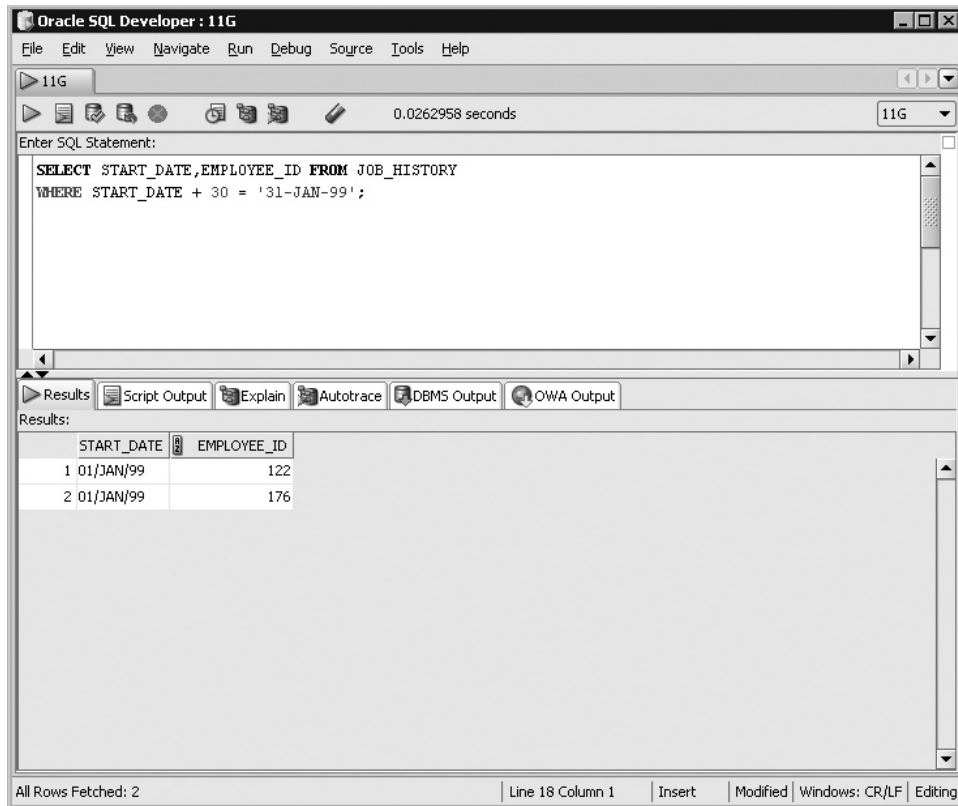
Arithmetic using the addition and subtraction operators is supported in expressions involving DATE values. An expression like: END_DATE - START_DATE returns a numeric value representing the number of days between START_DATE and END_DATE. An expression like: START_DATE + 30 returns a DATE value that is 30 days later than START_DATE. So the following expression is legitimate, as shown in Figure 3-6:

```
select employee_id from job_history
where start_date + 30 = '31-JAN-99';
```

This query returns rows from the JOB_HISTORY table containing a START_DATE value equal to 30 days before 31-JAN-1999. Therefore, only rows with a value of 01-JAN-1999 in the START_DATE column will be retrieved.

FIGURE 3-6

Using the
WHERE clause
with numeric
expressions



exam**Watch**

Conditional clauses compare two terms using comparison operators. It is important to understand the data types of the terms involved so they can be enclosed in single quotes, if necessary. A common mistake is to assume that a WHERE clause is syntactically correct, when in fact, it is missing quotation

marks that delimit character or date literals. Another common oversight is not being aware that the terms to the left and right of the comparison operator in a conditional clause may be expressions, columns, or literal values. Both these concepts may be tested in the exam.

Comparison Operators

The *equality* operator is used extensively to illustrate the concept of restricting rows using a WHERE clause. There are several alternative operators that may also be used. The *inequality* operators like “less than” or “greater than or equal to” may be used to return rows conforming to inequality conditions. The *BETWEEN* operator facilitates range-based comparison to test whether a column value lies between two values. The *IN* operator tests set membership, so a row is returned if the column value tested in the condition is a member of a set of literals. The pattern matching comparison operator *LIKE* is extremely powerful, allowing components of character column data to be matched to literals conforming to a specific pattern. The last comparison operator discussed in this section is the *IS NULL* operator, which returns rows where the column value contains a null value. These operators may be used in any combination in the WHERE clause and will be discussed next.

Equality and Inequality

Limiting the rows returned by a query involves specifying a suitable WHERE clause. If the clause is too restrictive, then few or no rows are returned. If the conditional clause is too broadly specified, then more rows than are required are returned. Exploring the different available operators should equip you with the language to request exactly those rows you are interested in. Testing for *equality* in a condition is both natural and intuitive. Such a condition is formed using the “is equal to” (=) operator. A row is returned if the equality condition is true for that row. Consider the following query:

```
select last_name, salary
from employees
where job_id='SA_REP';
```

The JOB_ID column of every row in the EMPLOYEES table is tested for equality with the character literal SA_REP. For character information to be equal, there must be an exact case-sensitive match. When such a match is encountered, the values for the projected columns, LAST_NAME and SALARY, are returned for that row, as shown in Figure 3-7. Note that although the conditional clause is based on the JOB_ID column, it is not necessary for this column to be projected by the query.

Inequality-based conditions enhance the WHERE clause specification. Range and pattern matching comparisons are possible using inequality and equality operators, but it is often preferable to use the BETWEEN and LIKE operators for these comparisons. The inequality operators are described in Table 3-1.

Inequality operators allow range-based queries to be fulfilled. You may be required to provide a set of results where a column value is *greater than* another value.

FIGURE 3-7

Conditions based on the equality operator

The screenshot shows the Oracle SQL Developer interface. The top pane displays the following SQL query:

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE JOB_ID='SA_REP';
```

The bottom pane shows the results of the query in a table format:

	LAST_NAME	SALARY
1	Tucker	10000
2	Bernstein	9500
3	Hall	9000
4	Olsen	8000
5	Cambrault	7500
6	Tuvault	7000
7	King	10000
8	Sully	9500
9	McEwen	9000
10	Smith	8000
11	Doran	7500
12	Sewall	7000

At the bottom of the window, it indicates "All Rows Fetched: 30".

TABLE 3-1Inequality
Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to
!=	Not equal to

For example, the following query may be issued to obtain a list of LAST_NAME and SALARY values for employees who earn more than \$5000:

```
select last_name, salary from employees
where salary > 5000;
```

Similarly, to obtain a list of employees who earn less than \$3000, the following query may be submitted:

```
select last_name, salary from employees
where salary < 3000;
```

The *composite inequality operators* (made up of more than one symbol) are utilized in the following four clauses:

```
Clause 1: where salary <= 3000;
Clause 2: where salary >= 5000;
Clause 3: where salary <> department_id;
Clause 4: where salary != 4000+department_id;
```

Clause 1 returns those rows which contain a SALARY value that is less than or equal to 3000. Clause 2 obtains data where the SALARY value is greater than or equal to 5000, whilst clauses 3 and 4 demonstrate the two forms of the “not equal to” operators. Clause 3 returns the rows which have SALARY column values that are not equal to the DEPARTMENT_ID values. The alternate “not equal to” operator in clause 4 illustrates that columns, literals, and expressions may all be compared using inequality operators. Clause 4 returns those rows which contain a SALARY value that is not equal to the sum of the DEPARTMENT_ID for that row and 4000.

Numeric inequality is naturally intuitive. The comparison of character and date terms, however, is more complex. Testing character inequality is interesting since the strings being compared on either side of the inequality operator are converted

to a numeric representation of its characters. Based on the database character set and NLS (National Language Support) settings, each character string is assigned a numeric value. These numeric values form the basis for the evaluation of the inequality comparison. Consider the following statement:

```
select last_name from employees
where last_name < 'King';
```

The character literal 'King' is converted to a numeric representation. Assuming a US7ASCII database character set with AMERICAN NLS settings, the literal 'King' is converted into a sum of its ordinal character values: K + i + n + g = (75+105+110+103=393). For each row in the EMPLOYEES table, the LAST_NAME column is similarly converted to a numeric value. If this value is less than 393, then the row is selected. The same process for comparing numeric data using the inequality operators applies to character data. The only difference is that character data is converted implicitly by the Oracle server to a numeric value based on certain database settings.

Inequality comparisons operating on date values follow a similar process to character data. The Oracle server stores dates in an internal numeric format, and these values are compared within the conditions. The second of June of a certain year occurs earlier than the third of June of the same year. Therefore, the numeric value of the date 02-JUN-2008 is less than the numeric value of the date 03-JUN-2008. Consider the following query:

```
select last_name from employees
where hire_date < '01-JAN-2000';
```

This query retrieves each employee record containing a HIRE_DATE value that is earlier than '01-JAN-2000'. Rows with employee HIRE_DATE=31-DEC-1999 will be returned, while rows with employee HIRE_DATE values later than the first of January 2000 will not be returned, as shown in Figure 3-8.



The WHERE clause is a fundamental extension to the SELECT statement and forms part of most queries. Although many comparison operators exist, the majority of conditions are based on comparing two terms using both the equality and the inequality operators.

Range Comparison with the BETWEEN Operator

The BETWEEN operator tests whether a column or expression value falls within a range of two boundary values. The item must be at least the same as the lower boundary value, or at most the same as the higher boundary value, or fall within the range, for the condition to be true.