CERTIFICATION OBJECTIVE 10.03

Update Rows in a Table

The UPDATE command changes column values in one or more existing rows in a single table. The basic syntax is the following:

UPDATE *table* SET *column*=*value* [,*column*=*value*…] [WHERE *condition*];

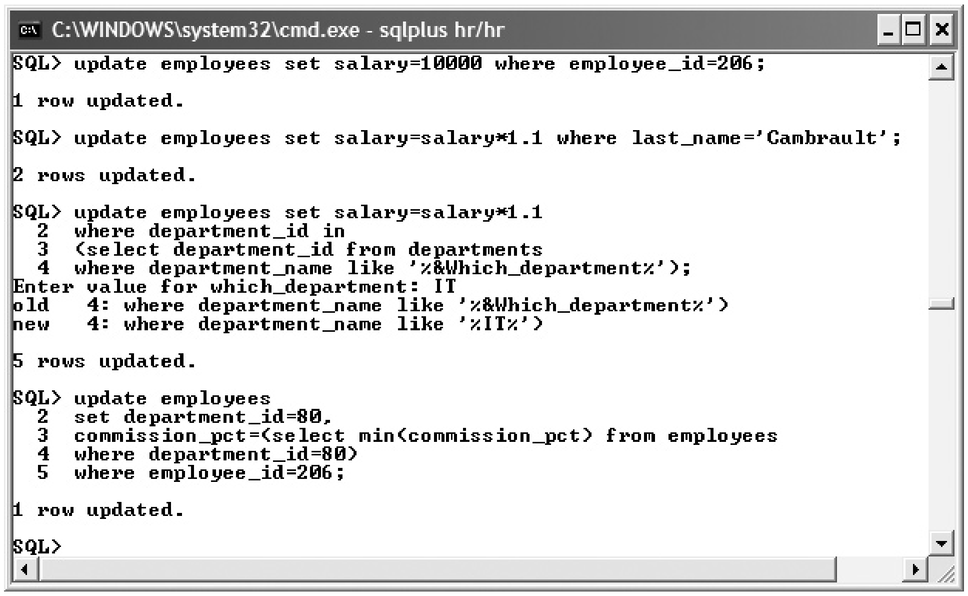
The more complex form of the command uses subqueries for one or more of the column values and for the WHERE condition. Figure 10-3 shows updates of varying complexity, executed from SQL*Plus.

The first example is the simplest. One column of one row is set to a literal value. Because the row is chosen with a WHERE clause that uses the equality predicate on the table's primary key, there is an absolute guarantee that at most only one row will be affected. No row will be changed if the WHERE clause fails to find any rows at all.

The second example shows use of arithmetic and an existing column to set the new value, and the row selection is not done on the primary key column. If the selection is not done on the primary key, or if a nonequality predicate (such as BETWEEN) is

FIGURE 10-3

Examples of using the UPDATE statement



```
SQL> update employees set salary=10000 where employee_id=206;

1 row updated.

SQL> update employees set salary=salary*1.1 where last_name='Cambrault';

2 rows updated.

SQL> update employees set salary=salary*1.1
  2  where department_id in
  3  (select department_id from departments
  4  where department_name like '%&Which_department%');
Enter value for which_department: IT
old   4: where department_name like '%&Which_department%')
new   4: where department_name like '%IT%')

5 rows updated.

SQL> update employees
  2  set department_id=80,
  3  commission_pct=(select min(commission_pct) from employees
  4  where department_id=80)
  5  where employee_id=206;

1 row updated.

SQL>
```

used, then the number of rows updated may be more than one. If the WHERE clause is omitted entirely, the update will be applied to every row in the table.

The third example in Figure 10-3 introduces the use of a subquery to define the set of rows to be updated. A minor additional complication is the use of a replacement variable to prompt the user for a value to use in the WHERE clause of the subquery. In this example, the subquery (lines 3 and 4) will select every employee who is in a department whose name includes the string 'IT' and increment their current salary by 10 percent (unlikely to happen in practice).

It is also possible to use subqueries to determine the value to which a column will be set, as in the fourth example. In this case, one employee (identified by primary key, in line 5) is transferred to department 80 (the sales department), and then the subquery in lines 3 and 4 set his commission rate to whatever the lowest commission rate in the department happens to be.

The syntax of an update that uses subqueries is as follows:

UPDATE *table*

SET *column*=[*subquery*] [,*column*=*subquery*…]

WHERE *column* = (*subquery*) [AND *column*=*subquery*…] ;

There is a rigid restriction on the subqueries using update columns in the SET clause: the subquery must return a *scalar* value. A scalar value is a single value of whatever data type is needed: the query must return one row, with one column. If the query returns several values, the UPDATE will fail. Consider these two examples:

```
update employees
set salary=(select salary from employees where employee_id=206);
update employees
set salary=(select salary from employees where last_name='Abel');
```

The first example, using an equality predicate on the primary key, will always succeed. Even if the subquery does not retrieve a row (as would be the case if there were no employee with EMPLOYEE_ID equal to 206), the query will still return a scalar value: a null. In that case, all the rows in EMPLOYEES would have their SALARY set to NULL—which might not be desired but is not an error as far as SQL is concerned. The second example uses an equality predicate on the LAST_NAME, which is not guaranteed to be unique. The statement will succeed if there is only one employee with that name, but if there were more than one it would fail with the error "ORA-01427: single-row subquery returns more than one row." For code that will work reliably, no matter what the state of the data, it is vital to ensure that the subqueries used for setting column values are scalar.

on the job

A common fix for making sure that queries are scalar is to use MAX or MIN. This version of the statement will always succeed:

```
update employees
set salary=(select max(salary) from employees where last_name='Abel');
```

However, just because it will work, doesn't necessarily mean that it does what is wanted.

The subqueries in the WHERE clause must also be scalar, if it is using the equality predicate (as in the preceding examples) or the greater/less than predicates. If it is using the IN predicate, then the query can return multiple rows, as in this example which uses IN:

```
update employees
set salary=10000
where department_id in (select department_id from departments
where department_name like '%IT%');
```

This will apply the update to all employees in a department whose name includes the string 'IT.' There are several of these. But even though the query can return several rows, it must still return only one column.

exam watch

The subqueries used to SET column values must be scalar subqueries. The subqueries used to select the rows must also be scalar, unless they use the IN predicate.

EXERCISE 10-2

Use the UPDATE Command

In this exercise, use various techniques to update rows in a table. It is assumed that the HR.REGIONS table is as seen in the illustration at the end of Exercise 10-1. If not, adjust the values as necessary.

1. Connect to the HR schema using SQL Developer or SQL*Plus.
2. Update a single row, identified by primary key:

```
update regions set region_name='Scandinavia' where region_id=101;
```

This statement should return the message “1 row updated.”

3. Update a set of rows, using a nonequality predicate:

```
update regions set region_name='Iberia' where region_id > 100;
```

This statement should return the message “3 rows updated.”

4. Update a set of rows, using subqueries to select the rows and to provide values:

```
update regions
set region_id=(region_id+(select max(region_id) from regions))
where region_id in (select region_id from regions where
region_id > 100);
```

This statement should return the message “3 rows updated.”

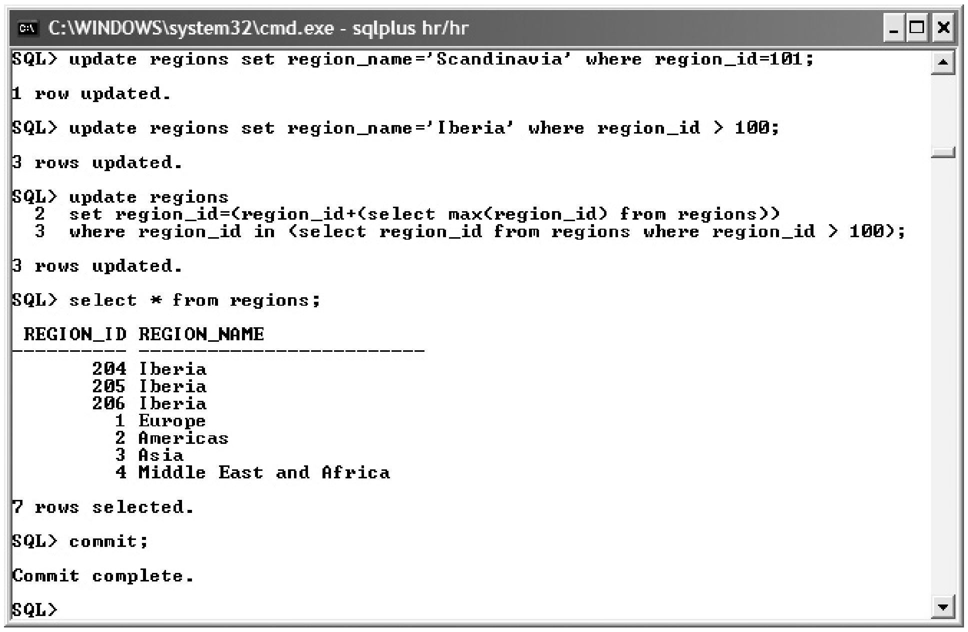
5. Confirm the state of the rows:

```
select * from regions;
```

6. Commit the changes made:

```
commit;
```

The following illustration shows the exercise, as done from SQL*Plus.



```
SQL> update regions set region_name='Scandinavia' where region_id=101;

1 row updated.

SQL> update regions set region_name='Iberia' where region_id > 100;

3 rows updated.

SQL> update regions
  2  set region_id=(region_id+(select max(region_id) from regions))
  3  where region_id in (select region_id from regions where region_id > 100);

3 rows updated.

SQL> select * from regions;

 REGION_ID REGION_NAME
---------- -------------------------
       204 Iberia
       205 Iberia
       206 Iberia
         1 Europe
         2 Americas
         3 Asia
         4 Middle East and Africa

7 rows selected.

SQL> commit;

Commit complete.

SQL>
```