Database Programming

Updating Column Values and Deleting Rows

Objectives

In this lesson, you will learn to:

- Construct and execute an UPDATE statement
- Construct and execute a DELETE statement
- Construct and execute a query that uses a subquery to update and delete data from a table
- Construct and execute a query that uses a correlated subquery to update and delete from a table
- Explain how foreign-key and primary-key integrity constraints affect UPDATE and DELETE statements
- Explain the purpose of the FOR UPDATE Clause in a SELECT statement

Purpose

Wouldn't it be a wonderful world if, once you got something done, it never needed to be changed or redone? Your bed would stay made, your clothes would stay clean, and you'd always be getting passing grades. Unfortunately in databases, as in life, "There is nothing permanent except change."

Updating, inserting, deleting, and managing data is a Database Administrator's (DBA's) job. In this lesson, you will become the DBA of your own schema and learn to manage your database.

UPDATE

The UPDATE statement is used to modify existing rows in a table. It requires four values:

- the name of the table
- the name of the column(s) whose values will be modified

UPDATE (cont.)

- a new value for each of the column(s) being modified
- a condition that identifies which rows in the table will be modified.

The new value for a column can be the result of a single-row subquery.

UPDATE (cont.)

The example shown uses an UPDATE statement to change the phone number of one customer in the Global Fast Foods database. Note that the copy_f_customers table is used in this transaction.

```
UPDATE copy_f_customers
  SET phone_number='4475582344'
  WHERE id=123;
```

ID	FIRST_NAME	LAST_NAME	ADDRESS	CITY	STATE	ZIP	PHONE_NO
123	Cole	Bee	123 Main Street	Orlando	FL	32838	4475582344
456	Zoe	Twee	1009 Oliver Avenue	Boston	MA	12889	7098675309
145	Katie	Hernandez	92 Chico Way	Los Angeles	CA	98008	8586667641

UPDATE (cont.)

We can change several columns and/or several rows in one UPDATE statement. This example changes both the phone number and the city for two Global Fast Foods customers.

```
UPDATE copy_f_customers
  SET phone_number='4475582344‘,
          city = ‘Chicago’
  WHERE id < 200;
```

ID	FIRST_NAME	LAST_NAME	ADDRESS	CITY	STATE	ZIP	PHONE_NO
123	Cole	Bee	123 Main Street	Chicago	FL	32838	4475582344
456	Zoe	Twee	1009 Oliver Avenue	Boston	MA	12889	7098675309
145	Katie	Hernandez	92 Chico Way	Chicago	CA	98008	4475582344

UPDATE (cont.)

Which rows would be updated in the following transaction?

```
UPDATE copy_f_customers
SET phone_number='9876543210';
```

ID	FIRST_NAME	LAST_NAME	ADDRESS	CITY	STATE	ZIP	PHONE_NO
123	Cole	Bee	123 Main Street	Orlando	FL	32838	4475582344
456	Zoe	Twee	1009 Oliver Avenue	Boston	MA	12889	7098675309
145	Katie	Hernandez	92 Chico Way	Los Angele	CA	98008	8586667641

Updating a Column with a value from a Subquery

We can use the result of a single-row subquery to provide the new value for an updated column.

```
UPDATE copy_f_staffs
SET salary  = (SELECT  salary
               FROM copy_f_staffs
               WHERE   id = 9)
WHERE id = 12;
```

This example changes the salary of one employee (id = 12) to the same salary as another employee (id = 9). As usual, the subquery executes first and retrieves the salary for employee id = 9. This salary value is then used to update the salary for employee id = 12.

ID	FIRST_NAME	LAST_NAME	SALARY
9	Bob	Miller	10
12	Sue	Doe	10

Updating Two Columns with Two Subquery Statements

To update several columns in one UPDATE statement, it is possible to write multiple single-row subqueries, one for each column.

In the following example the UPDATE statement changes the salary and staff type of one employee (id = 12) to the same values as another employee (id = 9).

Updating Two Columns with Two Subquery Statements (cont.)

```
UPDATE  copy_f_staffs
SET salary         =  (SELECT  salary
                       FROM   copy_f_staffs
                       WHERE   id = 9),
       staff_type  = (SELECT  staff_type
                      FROM   copy_f_staffs
                      WHERE   id = 9)
WHERE id  = 12;
```

ID	FIRST_NAME	LAST_NAME	SALARY	STAFF_TYPE
9	Bob	Miller	10	Cook
12	Sue	Doe	10	Cook

Updating Rows Based On Another Table

As you may have expected, the subquery can retrieve information from one table which is then used to update another table.

In this example, a copy of the f_staffs table was created. Then data from the original f_staffs table was retrieved, copied, and used to populate the copy of the f_staffs table.

```
UPDATE  copy_f_staffs
SET salary  = (SELECT salary
                        FROM f_staffs
                        WHERE id = 9)
WHERE id  = 9;
```

Updating Rows Based On The Same Table

As you already know subqueries can be either stand alone or correlated. In a correlated subquery, you are updating a row in a table based on a select from that same table.

Updating Rows Based On The Same Table (cont.)

In the example below, a copy of the department name column was created in the employees table. Then data from the original departments table was retrieved, copied, and used to populate the copy of the column in the employees table

```
UPDATE  employees e
SET e.department_name  = (SELECT d.department_name
                          FROM departments d
                          WHERE e.department_id = d.department_id);
```

DELETE

The DELETE statement is used to remove existing rows in a table. The statement requires two values:

- the name of the table
- the condition that identifies the rows to be deleted

DELETE (cont.)

The example shown uses the Global Fast Foods database to delete one row—the customer with ID number 123.

```
DELETE FROM copy_f_customers
WHERE ID= 123;
```

What do you predict will be deleted if the WHERE clause is eliminated in a DELETE statement?

All rows in the table are deleted if you omit the WHERE clause.

Subquery DELETE

Subqueries can also be used in DELETE statements.

The example shown deletes rows from the employees table for all employees who work in the Shipping department. Maybe this department has been renamed or eliminated.

```
DELETE FROM employees
WHERE  department_id =
            (SELECT department_id
            FROM   departments
            WHERE  department_name = 'Shipping');
```

Subquery DELETE

The example below deletes rows of all employees who work for a manager that manages more than 2 departments.

```
DELETE FROM employees e
WHERE  e.manager_id =
         (SELECT d.manager_id
         FROM     departments d
         HAVING  count (d.department_id) > 2
         GROUP BY d.manager_id);
```

Integrity Constraint Errors

Integrity constraints ensure that the data conforms to a needed set of rules. The constraints are automatically checked whenever a DML statement which could break the rules is executed. If any rule would be broken, the table is not updated and an error is returned.

Integrity Constraint Errors (cont.)

This example violates a NOT NULL constraint because first_name has a not null constraint and id=123 does not exist, so the subquery returns a null result.

```
UPDATE copy_f_staffs
SET first_name =  (SELECT first_name
                   FROM copy_f_staffs
                   WHERE id = 123);
```

ORA-01407: cannot update ("USQA_JOHN_SQL01_S01"."COPY_F_STAFFS"."FIRST_NAME") to NULL

Integrity Constraint Errors (cont.)

When will primary key - foreign key constraints be checked?

The EMPLOYEES table has a foreign key constraint on department_id which references the department_id of the DEPARTMENTS table. This ensures that every employee belongs to a valid department. In the DEPARTMENTS table, department_ids 10 and 20 exist but 15 does not.

Integrity Constraint Errors (cont.)

Which of the following statements will return an error?

```
1.      UPDATE employees SET department_id = 15
        WHERE employee_id = 100;

2.      DELETE FROM departments WHERE department_id = 10;

3.      UPDATE employees SET department_id = 10
        WHERE     department_id = 20;
```

Integrity Constraint Errors (cont.)

When modifying your table copies (for example copy_f_customers), you might see not null constraint errors, but you will not see any primary key – foreign key constraint errors.

The reason for this is that the CREATE TABLE …. AS (SELECT …) statement that is used to create the copy of the table, copies both the rows and the not null constraints, but does not copy the primary key – foreign key constraints.

Integrity Constraint Errors (cont.)

Therefore, no primary key – foreign key constraints exist on any of the copied tables.

Adding constraints is covered in another lesson.

FOR UPDATE Clause in a SELECT Statement

When a SELECT statement is issued against a database table, no locks are issued in the database on the rows returned by the query you are issuing. Most of the time this is how you want the database to behave—to keep the number of locks issued to a minimum. Sometimes, however, you want to make sure no one else can update or delete the records your query is returning while you are working on those records.

FOR UPDATE Clause in a SELECT Statement (cont.)

This is when the FOR UPDATE clause is used. As soon as your query is executed, the database will automatically issue exclusive row-level locks on all rows returned by your SELECT statement, which will be held until you issue a COMMIT or ROLLBACK command.

Reminder: APEX will autocommit, so of you want to test this example, remember to uncheck the Autocommit Box.

FOR UPDATE Clause in a SELECT Statement (cont.)

If you use a FOR UPDATE clause in a SELECT statement with multiple tables in it, all the rows from all tables will be locked.

```
SELECT e.employee_id, e.salary, d.department_name
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

FOR UPDATE Clause in a SELECT Statement (cont.)

These four rows are now locked by the user who ran the SELECT statement until the user issues a COMMIT or ROLLBACK.

EMPLOYEE_ID	SALARY	DEPARTMENT_NAME
141	3500	Shipping
142	3100	Shipping
143	2600	Shipping
144	2500	Shipping

Terminology

Key terms used in this lesson included:

- DELETE
- Integrity constraint
- UPDATE
- Correlated subquery UPDATE
- Correlated subquery DELETE
- FOR UPDATE of SELECT

Summary

In this lesson, you should have learned how to:

- Construct and execute an UPDATE statement
- Construct and execute a DELETE statement
- Construct and execute a query that uses a subquery to update and delete data from a table
- Construct and execute a query that uses a correlated subquery to update and delete from a table
- Explain how foreign-key and primary-key integrity constraints affect UPDATE and DELETE statements
- Explain the purpose of the FOR UPDATE Clause in a SELECT statement