

Database Programming

DEFAULT Values, MERGE, and Multi-Table Inserts



ACADEMY

Objectives

In this lesson, you will learn to:

- Understand when to specify a DEFAULT value
- Construct and execute a MERGE statement
- Construct and execute DML statements using subqueries
- Construct and execute multi-table inserts

Purpose

Up to now, you have been updating data using a single INSERT statement. It has been relatively easy when adding records one at a time, but what if your company is very large and utilizes a data warehouse to store sales records, customer, payroll, accounting, and personal data?

In this case, data is probably coming in from multiple sources and being managed by multiple people. Managing data one record at a time could be very confusing and very time consuming.

Purpose (cont.)

How do you determine what has been newly inserted or what has been recently changed?

In this lesson, you will learn a more efficient method to update and insert data using a sequence of conditional INSERT and UPDATE commands in a single atomic statement.

Purpose (cont.)

You will also learn how to retrieve data from one single subquery and INSERT the rows returned into more than one target table.

As you extend your knowledge in SQL, you'll appreciate effective ways to accomplish your work.

DEFAULT

Each column in a table can have a default value specified for it. In the event that a new row is inserted and no value for the column is assigned, the default value will be assigned instead of a null value.

DEFAULT (cont.)

Using default values allows you to control where and when the default value should be applied. The default value can be a literal value, an expression, or a SQL function such as SYSDATE and USER, but the value cannot be the name of another column.

The default value must match the data type of the column. DEFAULT can be specified for a column when the table is created or altered.

DEFAULT Example

The example below shows a default value being specified for the hire_date column at the time the table is created:

```
CREATE TABLE my_employees (  
  hire_date          DATE DEFAULT SYSDATE,  
  first_name         VARCHAR2(15),  
  last_name          VARCHAR2(15));
```

When rows are added to this table, SYSDATE will be assigned to any row that does not explicitly specify a hire_date value.

Explicit DEFAULT with INSERT

Explicit defaults can be used in INSERT and UPDATE statements. The INSERT example using the DEPARTMENTS table shows the explicit use of DEFAULT.

```
INSERT INTO departments  
  (department_id, department_name, manager_id)  
VALUES  
  (300, 'Engineering', DEFAULT);
```

If a default value was specified for the manager_id column when the table was created, Oracle assigns the default value to the column. However, if no default value was specified, Oracle assigns a null value.

Explicit DEFAULT with UPDATE

Explicit defaults can be used in INSERT and UPDATE statements. The UPDATE example using the DEPARTMENTS table shows explicit use of DEFAULT.

```
UPDATE departments  
SET manager_id = DEFAULT  
WHERE department_id = 10;
```

If a default value was specified for the department_id column, the column is assigned the default value. However, if no default value was specified when the column was created, a null value is assigned.

MERGE

Using the MERGE statement accomplishes two tasks at the same time. MERGE will INSERT and UPDATE simultaneously. If a value is missing, a new one is inserted. If a value exists but needs to be changed, MERGE will update it.

To perform these kinds of changes to database tables, you need to have INSERT and UPDATE privileges on the target table and SELECT privileges on the source table. Aliases can be used with the MERGE statement.

MERGE Syntax

One row at a time is read from the source table and compared to rows in the destination table using the matching condition. If a matching row exists in the destination table, the source row is used to update one or more columns in the matching destination row. If a matching row does not exist, values from the source row are used to insert a new row into the destination table.

```
MERGE INTO destination-table USING source-table  
ON matching-condition  
WHEN MATCHED THEN UPDATE  
SET .....  
WHEN NOT MATCHED THEN INSERT  
VALUES (.....);
```

MERGE Example

This example uses the EMPLOYEES table (alias e) as a data source to insert and update rows in a copy of the table named COPY_EMP (alias c).

```
MERGE INTO copy_emp c USING employees e
  ON (c.employee_id = e.employee_id)
  WHEN MATCHED THEN UPDATE
    SET
      c.last_name      = e.last_name,
      c.department_id  = e.department_id
  WHEN NOT MATCHED THEN INSERT
    VALUES (e.employee_id, e.last_name, e.department_id);
```

MERGE Example (cont.)

EMPLOYEES rows 100 and 103 have matching rows in COPY_EMP, and so the matching COPY_EMP rows were updated.

EMPLOYEE 142 had no matching row, and so was inserted into COPY_EMP.

MERGE Example (cont.)

EMPLOYEES (source table)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
103	Hunold	60
142	Davies	50

COPY_EMP before the MERGE is executed

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	Smith	40
103	Chang	30

COPY_EMP after the MERGE has executed

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
103	Hunold	60
142	Davies	50

Multi-Table Inserts

Multi-table inserts are used when the same source data should be inserted into more than one target table. This functionality is useful when you are working in a data warehouse environment where it is common to regularly move data from the operational systems into a data warehouse for analytical reporting and analysis. Creating and managing data warehouses is one way of managing the sometimes very high number of rows inserted into operational systems during a normal working day.

Multi-Table Inserts (cont.)

Imagine, for instance, how many rows of data your telephone provider must create daily for all of your calls or text messages made on all of the devices that you have access to?

Then add to that your internet surfing and downloads of ringtones, wallpapers, games, and other mobile applications.

Multi-Table Inserts (cont.)

Multiply that number by the total number of customers, and that might give you an idea of the amount of data the telecommunication companies have to manage.

These rows may have to be inserted into more than one table in the data warehouse, so if we can just `SELECT` them once and then replicate them, that will improve the performance.

Multi-Table Inserts (cont.)

Multi-table inserts can be unconditional or conditional. In an unconditional multi-table insert, Oracle will insert all rows returned by the subquery into all table insert clauses found in the statement.

In a conditional multi-table insert, you can specify either **ALL** or **FIRST**.

Multi-Table Inserts (cont.)

ALL

If you specify ALL, the default value, the database evaluates each WHEN clause regardless of the results of the evaluation of any other WHEN clause.

For each WHEN clause whose condition evaluates to true, the database executes the corresponding INTO clause list.

Multi-Table Inserts (cont.)

FIRST

If you specify FIRST, the database evaluates each WHEN clause in the order in which it appears in the statement. For the first WHEN clause that evaluates to true, the database executes the corresponding INTO clause and skips subsequent WHEN clauses for the given row.

Multi-Table Inserts (cont.)

ELSE clause

For a given row, if no WHEN clause evaluates to true, the database executes the INTO clause list associated with the ELSE clause.

If you did not specify an else clause, then the database takes no action for that row.

Multi-Table Inserts Syntax

Multi-table insert statement syntax is as follows:

```
INSERT ALL INTO clause VALUES clause SUBQUERY
```

Multi-table insert statement example is as follows:

```
INSERT ALL
                                INTO all_calls VALUES (caller_id,
call_timestamp, call_duration, call_format)
                                INTO police_record_calls VALUES
(call_timestamp, recipient_caller)
      SELECT  caller_id, call_timestamp, call_duration, call_format ,
recipient_caller)
      FROM      calls
      WHERE      TRUNC(call_timestamp ) = TRUNC(SYSDATE )
```

Multi-Table Inserts Conditional

```
INSERT ALL
    WHEN call_format IN ('tlk','txt','pic') THEN
        INTO all_calls VALUES (caller_id, call_timestamp,
call_duration, call_format)
    WHEN call_format IN ('tlk','txt') THEN
        INTO police_record_calls VALUES (caller_id,
call_timestamp, recipient_caller)
    WHEN call_duration < 50 AND call_type = 'tlk' THEN
        INTO short_calls VALUES (caller_id, call_timestamp,
call_duration)
    WHEN call_duration >= 50 AND call_type = 'tlk' THEN
        INTO long_calls VALUES (caller_id, call_timestamp,
call_duration)
    SELECT caller_id, call_timestamp, call_duration, call_format ,
recipient_caller)
    FROM calls
    WHERE TRUNC(call_timestamp ) = TRUNC(SYSDATE )
```


Terminology

Key terms used in this lesson included:

- DEFAULT
- MERGE
- Multi-Table Inserts
- ALL, FIRST, and ELSE

Summary

In this lesson, you should have learned how to:

- Understand when to specify a DEFAULT value
- Construct and execute a MERGE statement
- Construct and execute DML statements using subqueries
- Construct and execute multi-table inserts