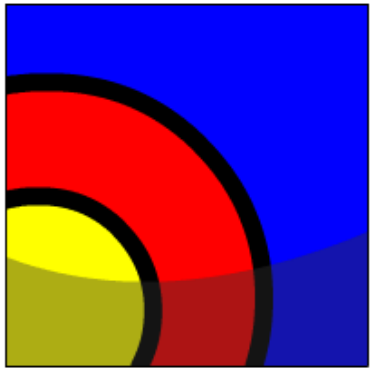
DEFAULT Values, MERGE, and Multi-Table Inserts

What Will I Learn?

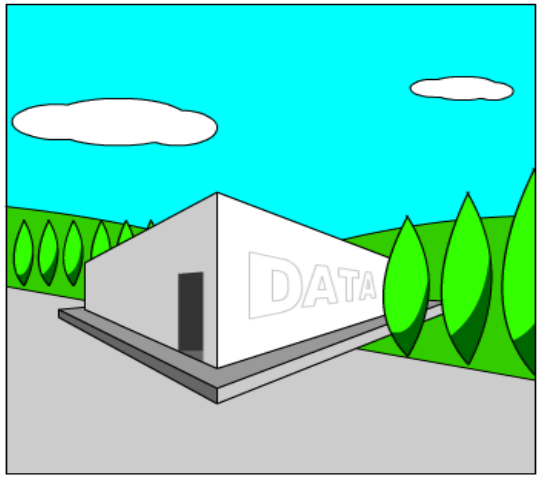
In this lesson, you will learn to:



- 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

Why Learn It?

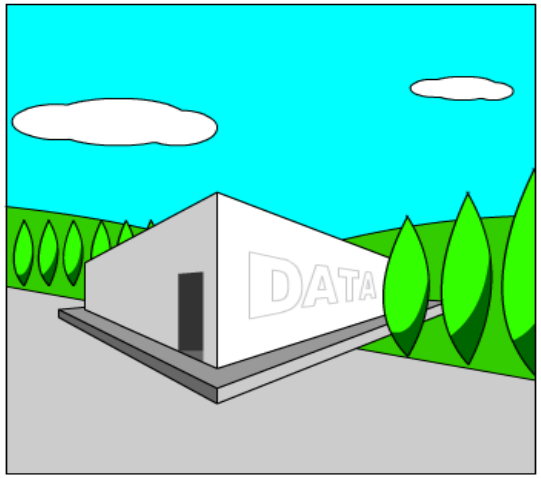
Up to now, you have been updating data using a single INSERT statement. It has been relatively easy when adding records one at a time, but what if your company is very large and utilizes a data warehouse to store sales records and customer, payroll, accounting, and personal data?



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.

Why Learn It?

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.

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.

Tell Me / Show Me

DEFAULT

A column in a table can be given a default value. This option prevents null values from entering the columns if a row is inserted without a specified value for the column.

Using default values also 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.

Tell Me / Show Me

The example below shows a default value being specified 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 added to any row that does not explicitly specify a hire_date value.

Tell Me / Show Me

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 set for the manager_id column, Oracle sets the column to the default value. However, if no default value was set when the column was created, Oracle inserts a null value.

Tell Me / Show Me

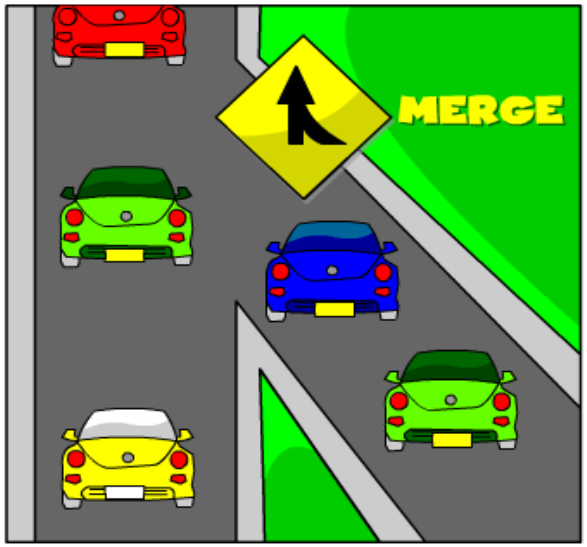
Explicit DEFAULT with UPDATE

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

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

If a default value was set for the department_id column, Oracle sets the column to the default value. However, if no default value was set when the column was created, Oracle inserts a null value.

Tell Me / Show Me



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.

Tell Me / Show Me

MERGE SYNTAX

MERGE INTO destination-table USING source-table

ON matching-condition

WHEN MATCHED THEN UPDATE

SET ……

WHEN NOT MATCHED THEN INSERT

VALUES (……);

One row at a time is read from the source table, and its column values are compared with 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 column(s) 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.

Tell Me / Show Me

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);
```

The next slide shows the result of this statement.

Tell Me / Show Me

```
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);
```

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.

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

Tell Me / Show Me

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 especially useful when you are working in a data warehouse environment, where it is very common to move data on a regular basis 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.

Imagine, for instance, how many rows your mobile/cell telephone provider must create daily.

At least one for each time you use your mobile/cell phone, and how many calls do you make and receive a day?

Then add the number of SMS's you send and receive.

Add to that your mobile surfing and downloads of ringtones, wallpapers, games and other mobile applications.

Multiply that number by the number of customers.

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.

Tell Me / Show Me

Multi-Table Inserts

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.

ALL

If you specify ALL, the default value, then 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.

FIRST

If you specify FIRST, then 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.

ELSE clause

For a given row, if no WHEN clause evaluates to true, then:

If you have specified an ELSE clause, then 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.

Tell Me / Show Me

Multi-Table Inserts

Multi-table insert statement syntax is as follows:

INSERT ALL INTO clause VALUES clause SUBQUERY

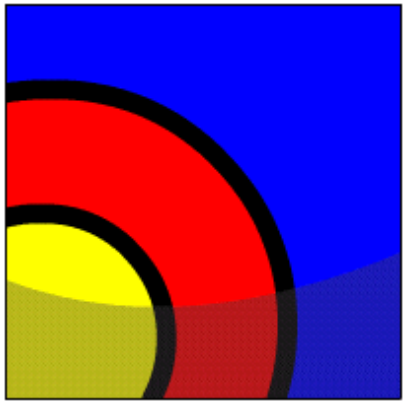
```
INSERT ALL
        INTO all_calls VALUES (caller_id, call_timestamp, call_duration,
call_format)
        INTO police_record_calls VALUES (caller_id, call_timestamp,
recipient_caller)
  SELECT   caller_id, call_timestamp, call_duration, call_format ,
recipient_caller)
  FROM     calls
  WHERE    TRUNC(call_timestamp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

Tell Me / Show Me

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 )
```

Summary



In this lesson you have learned 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

Summary

Practice Guide

The link for the lesson practice guide can be found in the course resources in Section 0.