7

Manipulating Data by Using Subqueries

Objectives

After completing this lesson, you should be able to:

- Use subqueries to manipulate data
- Insert values by using a subquery as a target
- Use the `WITH CHECK OPTION` keyword on DML statements
- Use correlated subqueries to update and delete rows

ORACLE

In this lesson, you learn how to manipulate data in the Oracle database by using subqueries. You also learn how to solve problems by using correlated subqueries.

Lesson Agenda

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

Using Subqueries to Manipulate Data

You can use subqueries in data manipulation language (DML) statements to:

- Retrieve data by using an inline view
- Copy data from one table to another
- Update data in one table based on the values of another table
- Delete rows from one table based on rows in another table

ORACLE

Subqueries can be used to retrieve data from a table that you can use as input to an `INSERT` into a different table. In this way, you can easily copy large volumes of data from one table to another with one single `SELECT` statement. Similarly, you can use subqueries to do mass updates and deletes by using them in the `WHERE` clause of the `UPDATE` and `DELETE` statements. You can also use subqueries in the `FROM` clause of a `SELECT` statement. This is called an inline view.

Note: You learned how to update and delete rows based on another table in the course titled *Oracle Database: SQL Workshop I*.

Lesson Agenda

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

ORACLE

Inserting by Using a Subquery as a Target

```
INSERT INTO (SELECT l.location_id, l.city, l.country_id
             FROM   loc l
             JOIN   countries c
             ON(l.country_id = c.country_id)
             JOIN regions USING(region_id)
             WHERE region_name = 'Europe')
VALUES (3300, 'Cardiff', 'UK');
```

```
1 rows inserted.
```

ORACLE

You can use a subquery in place of the table name in the `INTO` clause of the `INSERT` statement. The `SELECT` list of this subquery must have the same number of columns as the column list of the `VALUES` clause. Any rules on the columns of the base table must be followed in order for the `INSERT` statement to work successfully. For example, you cannot put in a duplicate location `ID` or leave out a value for a mandatory `NOT NULL` column.

This use of subqueries helps you avoid having to create a view just for performing an `INSERT`.

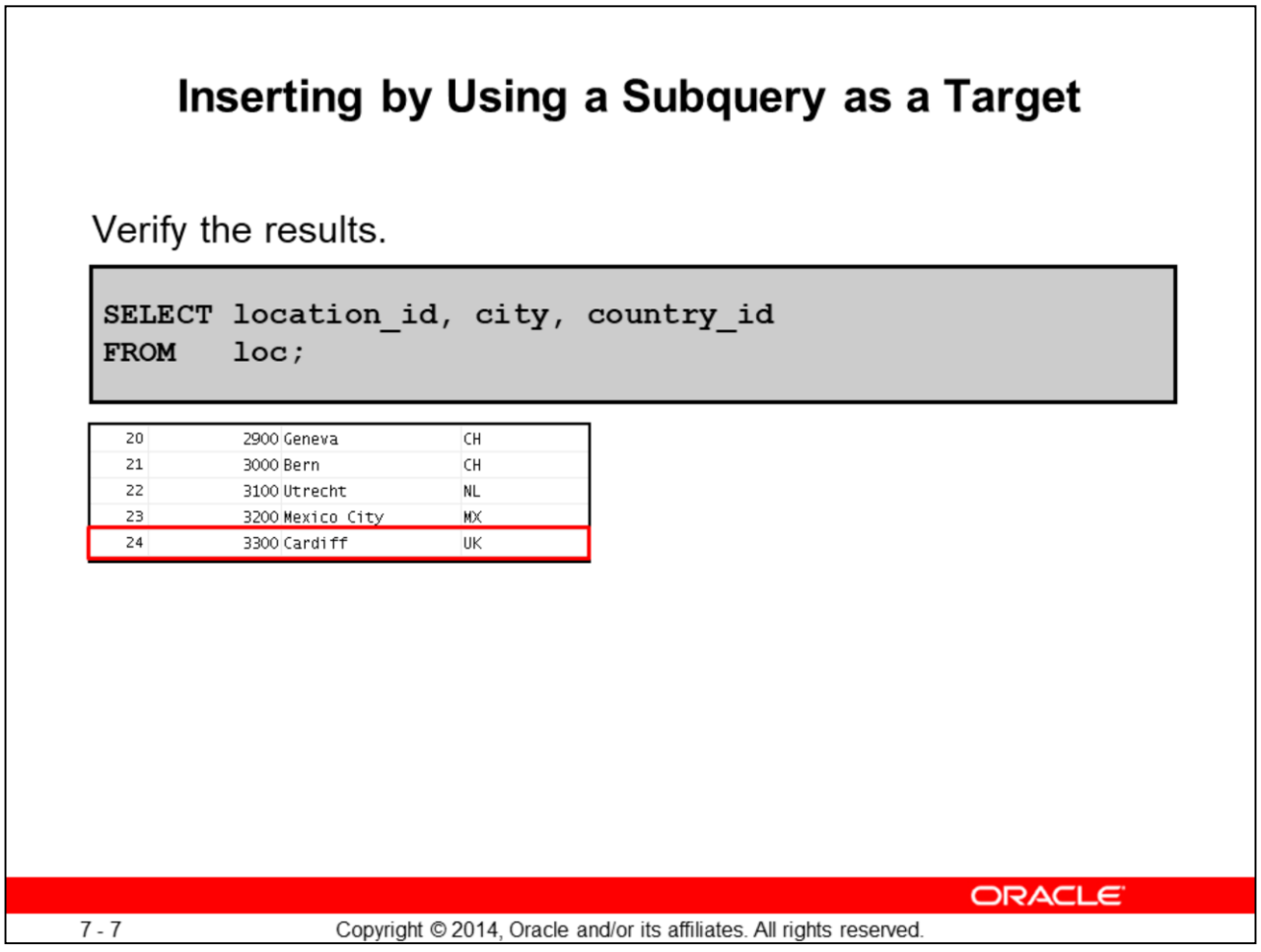
The example in the slide uses a subquery in the place of `LOC` to create a record for a new European city.

Note: You can also perform the `INSERT` operation on the `EUROPEAN_CITIES` view by using the following code:

```
INSERT INTO european_cities
VALUES (3300,'Cardiff','UK');
```

For the example in the slide, the loc table is created by running the following statement:

```
CREATE TABLE loc AS SELECT * FROM locations;
```



The example in the slide shows that the insert via the inline view created a new record in the base table `LOC`.

The following example shows the results of the subquery that was used to identify the table for the `INSERT` statement.

```
SELECT l.location_id, l.city, l.country_id
FROM   loc l
JOIN   countries c
ON(l.country_id = c.country_id)
JOIN regions USING(region_id)
WHERE region_name = 'Europe';
```

Lesson Agenda

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

ORACLE

Using the WITH CHECK OPTION Keyword on DML Statements

The WITH CHECK OPTION keyword prohibits you from changing rows that are not in the subquery.

```
INSERT INTO ( SELECT location_id, city, country_id
              FROM   loc
              WHERE  country_id IN
              (SELECT country_id
               FROM countries
               NATURAL JOIN regions
               WHERE region_name = 'Europe')
              WITH CHECK OPTION )
VALUES (3600, 'Washington', 'US');
```

```
Error report:
SQL Error: ORA-01402: view WITH CHECK OPTION where-clause violation
01402. 00000 - "view WITH CHECK OPTION where-clause violation"
*Cause:
*Action:
```

ORACLE

Specify the WITH CHECK OPTION keyword to indicate that if the subquery is used in place of a table in an INSERT, UPDATE, or DELETE statement, no changes that will produce rows that are not included in the subquery are permitted to that table.

The example in the slide shows how to use an inline view with WITH CHECK OPTION. The INSERT statement prevents the creation of records in the LOC table for a city that is not in Europe.

The following example executes successfully because of the changes in the VALUES list.

```
INSERT INTO (SELECT location_id, city, country_id
             FROM   loc
             WHERE  country_id IN
             (SELECT country_id
              FROM countries
              NATURAL JOIN regions
              WHERE region_name = 'Europe')
              WITH CHECK OPTION)
VALUES (3500, 'Berlin', 'DE');
```

The use of an inline view with the `WITH CHECK OPTION` provides an easy method to prevent changes to the table.

To prevent the creation of a non-European city, you can also use a database view by performing the following steps:

1. Create a database view:

```
CREATE OR REPLACE VIEW european_cities
AS
SELECT location_id, city, country_id
FROM   locations
WHERE  country_id in
  (SELECT country_id
   FROM countries
   NATURAL JOIN regions
   WHERE region_name = 'Europe')
WITH CHECK OPTION;
```

2. Verify the results by inserting data:

```
INSERT INTO european_cities
VALUES (3400,'New York','US');
```

The second step produces the same error as shown in the slide.

Lesson Agenda

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

ORACLE

Correlated UPDATE

Use a correlated subquery to update rows in one table based on rows from another table.

```
UPDATE table1 alias1
SET    column = (SELECT expression
                 FROM   table2 alias2
                 WHERE  alias1.column =
                        alias2.column);
```

ORACLE

In the case of the UPDATE statement, you can use a correlated subquery to update rows in one table based on rows from another table.

Using Correlated UPDATE

- Denormalize the EMPL6 table by adding a column to store the department name.
- Populate the table by using a correlated update.

```
ALTER TABLE empl6
ADD(department_name VARCHAR2(25));
```

```
UPDATE empl6 e
SET    department_name =
              (SELECT department_name
              FROM   departments d
              WHERE  e.department_id = d.department_id);
```

ORACLE

The example in the slide denormalizes the EMPL6 table by adding a column to store the department name and then populates the table by using a correlated update.

Following is another example for a correlated update.

Problem Statement

The REWARDS table has a list of employees who have exceeded expectations in their performance. Use a correlated subquery to update rows in the EMPL6 table based on rows from the REWARDS table:

```
UPDATE empl6
SET    salary = (SELECT empl6.salary + rewards.pay_raise
                 FROM   rewards
                 WHERE  employee_id  =
                        empl6.employee_id
                 AND  payraise_date =
                      (SELECT MAX(payraise_date)
                      FROM   rewards
                      WHERE  employee_id = empl6.employee_id))
WHERE  empl6.employee_id
IN     (SELECT employee_id FROM rewards);
```

This example uses the REWARDS table. The REWARDS table has the following columns: EMPLOYEE_ID, PAY_RAISE, and PAYRAISE_DATE. Every time an employee gets a pay raise, a record with details such as the employee ID, the amount of the pay raise, and the date of receipt of the pay raise is inserted into the REWARDS table. The REWARDS table can contain more than one record for an employee. The PAYRAISE _DATE column is used to identify the most recent pay raise received by an employee.

In the example, the SALARY column in the EMPL6 table is updated to reflect the latest pay raise received by the employee. This is done by adding the current salary of the employee with the corresponding pay raise from the REWARDS table.

Correlated DELETE

Use a correlated subquery to delete rows in one table based on rows from another table.

```
DELETE FROM table1 alias1
WHERE  column operator
               (SELECT expression
                FROM   table2 alias2
                WHERE  alias1.column = alias2.column);
```

ORACLE

In the case of a DELETE statement, you can use a correlated subquery to delete only those rows that also exist in another table. If you decide that you will maintain only the last four job history records in the JOB_HISTORY table, when an employee transfers to a fifth job, you delete the oldest JOB_HISTORY row by looking up the JOB_HISTORY table for the MIN(START_DATE) for the employee. The following code illustrates how the preceding operation can be performed using a correlated DELETE:

```
DELETE FROM job_history JH
WHERE employee_id =
      (SELECT employee_id
       FROM employees E
       WHERE JH.employee_id = E.employee_id
       AND START_DATE =
             (SELECT MIN(start_date)
              FROM job_history JH
              WHERE JH.employee_id = E.employee_id)
              AND 5 >  (SELECT COUNT(*)
                        FROM job_history JH
                        WHERE JH.employee_id = E.employee_id
                        GROUP BY EMPLOYEE_ID
```

```
                                HAVING COUNT(*) >= 4));
```

Using Correlated DELETE

Use a correlated subquery to delete only those rows from the EMPL6 table that also exist in the EMP_HISTORY table.

```
DELETE FROM empl6 E
WHERE employee_id =
           (SELECT employee_id
            FROM   emp_history
            WHERE  employee_id = E.employee_id);
```

ORACLE

Example

Two tables are used in this example. They are:

- The EMPL6 table, which provides details of all the current employees
- The EMP_HISTORY table, which provides details of previous employees

EMP_HISTORY contains data regarding previous employees, so it would be erroneous if the same employee's record existed in both the EMPL6 and EMP_HISTORY tables. You can delete such erroneous records by using the correlated subquery shown in the slide.

Summary

In this lesson, you should have learned how to:

- Manipulate data by using subqueries
- Insert values by using a subquery as a target
- Use the `WITH CHECK OPTION` keyword on DML statements
- Use correlated subqueries with `UPDATE` and `DELETE` statements

ORACLE

In this lesson, you should have learned how to manipulate data in the Oracle database by using subqueries. You learn how to use the `WITH CHECK OPTION` keyword on DML statements and use correlated subqueries with `UPDATE` and `DELETE` statements.

Practice 7: Overview

This practice covers the following topics:

- Using subqueries to manipulate data
- Inserting values by using a subquery as a target
- Using the `WITH CHECK OPTION` keyword on DML statements
- Using correlated subqueries to update and delete rows

ORACLE

In this practice, you learn the concepts of manipulating data by using subqueries, `WITH CHECK OPTION`, and correlated subqueries to `UPDATE` and `DELETE` rows.