3

Interacting with the Oracle Server

Objectives

After completing this lesson, you should be able to do the following:

- Write a successful SELECT statement in PL/SQL
- Write DML statements in PL/SQL
- Control transactions in PL/SQL
- Determine the outcome of SQL data manipulation language (DML) statements

SQL Statements in PL/SQL

- Extract a row of data from the database by using the `SELECT` command.
- Make changes to rows in the database by using DML commands.
- Control a transaction with the `COMMIT`, `ROLLBACK`, or `SAVEPOINT` command.
- Determine DML outcome with implicit cursor attributes.

SELECT Statements in PL/SQL

Retrieve data from the database with a SELECT statement.

Syntax:

```
SELECT  select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
[WHERE  condition];
```

SELECT Statements in PL/SQL

- The `INTO` clause is required.
- Queries must return one and only one row.

Example:

```
DECLARE
  v_deptno           NUMBER(4);
  v_location_id      NUMBER(4);
BEGIN
  SELECT      department_id, location_id
  INTO        v_deptno, v_location_id
  FROM        departments
  WHERE       department_name = 'Sales';
  ...
END;
/
```

Retrieving Data in PL/SQL

Retrieve the hire date and the salary for the specified employee.

Example:

```
DECLARE
  v_hire_date   employees.hire_date%TYPE;
  v_salary      employees.salary%TYPE;
BEGIN
  SELECT   hire_date, salary
  INTO     v_hire_date, v_salary
  FROM     employees
  WHERE    employee_id = 100;
  ...
END;
/
```

Retrieving Data in PL/SQL

Return the sum of the salaries for all employees in the specified department.

Example:

```
SET SERVEROUTPUT ON
DECLARE
  v_sum_sal   NUMBER(10,2);
  v_deptno     NUMBER NOT NULL := 60;
BEGIN
  SELECT      SUM(salary)  -- group function
  INTO        v_sum_sal
  FROM        employees
  WHERE       department_id = v_deptno;
  DBMS_OUTPUT.PUT_LINE ('The sum salary is ' ||
                        TO_CHAR(v_sum_sal));
END;
/
```

Naming Conventions

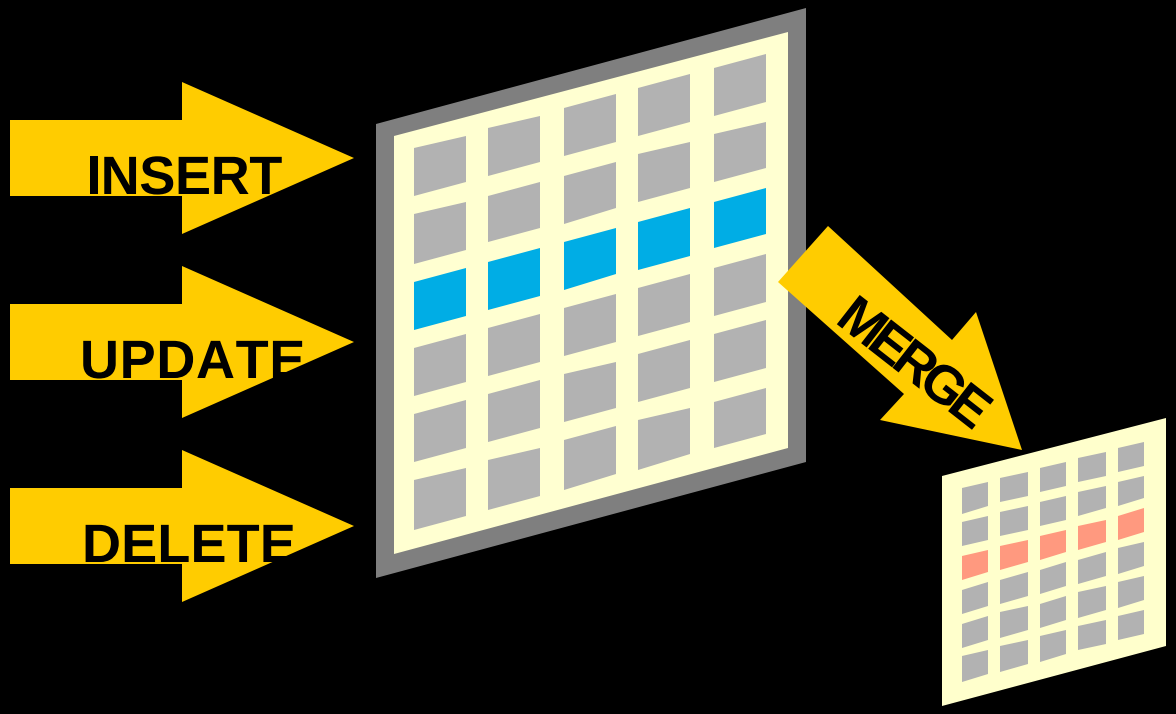
```
DECLARE
  hire_date       employees.hire_date%TYPE;
  sysdate         hire_date%TYPE;
  employee_id     employees.employee_id%TYPE := 176;
BEGIN
  SELECT      hire_date, sysdate
  INTO        hire_date, sysdate
  FROM        employees
  WHERE       employee_id = employee_id;
END;
/
```

```
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
```

Manipulating Data Using PL/SQL

Make changes to database tables by using DML commands:

- `INSERT`
- `UPDATE`
- `DELETE`
- `MERGE`



Inserting Data

Add new employee information to the EMPLOYEES table.

Example:

```
BEGIN
  INSERT INTO employees
  (employee_id, first_name, last_name, email,
   hire_date, job_id, salary)
  VALUES
   (employees_seq.NEXTVAL, 'Ruth', 'Cores', 'RCORES',
    sysdate, 'AD_ASST', 4000);
END;
/
```

Updating Data

Increase the salary of all employees who are stock clerks.

Example:

```
DECLARE
  v_sal_increase   employees.salary%TYPE := 800;
BEGIN
  UPDATE      employees
  SET         salary = salary + v_sal_increase
  WHERE       job_id = 'ST_CLERK';
END;
/
```

Deleting Data

Delete rows that belong to department 10 from the EMPLOYEES table.

Example:

```
DECLARE
  v_deptno   employees.department_id%TYPE := 10;
BEGIN
  DELETE FROM   employees
  WHERE         department_id = v_deptno;
END;
/
```

Merging Rows

Insert or update rows in the COPY_EMP table to match the EMPLOYEES table.

```
DECLARE
      v_empno employees.employee_id%TYPE := 100;
BEGIN
MERGE INTO copy_emp c
     USING employees e
     ON (e.employee_id = v_empno)
   WHEN MATCHED THEN
     UPDATE SET
       c.first_name     = e.first_name,
       c.last_name      = e.last_name,
       c.email          = e.email,
       . . .
   WHEN NOT MATCHED THEN
     INSERT VALUES(e.employee_id, e.first_name, e.last_name,
          . . .,e.department_id);
END;
```

Naming Conventions

- Use a naming convention to avoid ambiguity in the `WHERE` clause.
- Database columns and identifiers should have distinct names.
- Syntax errors can arise because PL/SQL checks the database first for a column in the table.
- The names of local variables and formal parameters take precedence over the names of database tables.
- The names of database table columns take precedence over the names of local variables.

SQL Cursor

- A cursor is a private SQL work area.
- There are two types of cursors:
  - Implicit cursors
  - Explicit cursors
- The Oracle server uses implicit cursors to parse and execute your SQL statements.
- Explicit cursors are explicitly declared by the programmer.

SQL Cursor Attributes

Using SQL cursor attributes, you can test the outcome of your SQL statements.

SQL%ROWCOUNT	**Number of rows affected by the most recent SQL statement (an integer value)**
SQL%FOUND	**Boolean attribute that evaluates to TRUE if the most recent SQL statement affects one or more rows**
SQL%NOTFOUND	**Boolean attribute that evaluates to TRUE if the most recent SQL statement does not affect any rows**
SQL%ISOPEN	**Always evaluates to FALSE because PL/SQL closes implicit cursors immediately after they are executed**

SQL Cursor Attributes

Delete rows that have the specified employee ID from the EMPLOYEES table. Print the number of rows deleted.

Example:

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
  v_employee_id employees.employee_id%TYPE := 176;
BEGIN
  DELETE FROM  employees
  WHERE        employee_id = v_employee_id;
  :rows_deleted := (SQL%ROWCOUNT ||
                        ' row deleted.');
END;
/
PRINT rows_deleted
```

Transaction Control Statements

- Initiate a transaction with the first DML command to follow a `COMMIT` or `ROLLBACK`.
- Use `COMMIT` and `ROLLBACK` SQL statements to terminate a transaction explicitly.

Summary

In this lesson you should have learned how to:

- Embed SQL in the PL/SQL block using `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `MERGE`
- Embed transaction control statements in a PL/SQL block `COMMIT`, `ROLLBACK`, and `SAVEPOINT`

Summary

In this lesson you should have learned that:

- There are two cursor types: implicit and explicit.
- Implicit cursor attributes are used to verify the outcome of DML statements:
  - SQL%ROWCOUNT
  - SQL%FOUND
  - SQL%NOTFOUND
  - SQL%ISOPEN
- Explicit cursors are defined by the programmer.

Practice 3

1. Create a PL/SQL block that selects the maximum department number in the `DEPARTMENTS` table and stores it in an *i*SQL*Plus variable. Print the results to the screen. Save your PL/SQL block in a file named `p3q1.sql`. by clicking the `Save Script` button. Save the script with a `.sql` extension.

G_MAX_DEPTNO
270

2. Modify the PL/SQL block you created in exercise 1 to insert a new department into the `DEPARTMENTS` table. Save the PL/SQL block in a file named `p3q2.sql` by clicking the `Save Script` button. Save the script with a `.sql` extension.
    a. Use the `DEFINE` command to provide the department name. Name the new department `Education`.
    b. Pass the value defined for the department name to the PL/SQL block through a *i*SQL*Plus substitution variable. Rather than printing the department number retrieved from exercise 1, add 10 to it and use it as the department number for the new department.
    c. Leave the location number as null for now.
    d. Execute the PL/SQL block.
    e. Display the new department that you created.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Education		

3. Create a PL/SQL block that updates the location ID for the new department that you added in the previous practice. Save your PL/SQL block in a file named `p3q3.sql` by clicking the `Save Script` button. Save the script with a `.sql` extension.
    a. Use an *i*SQL*Plus variable for the department ID number that you added in the previous practice.
    b. Use the `DEFINE` command to provide the location ID. Name the new location ID 1700.

```
DEFINE p_deptno = 280
DEFINE p_loc = 1700
```

    c. Pass the value to the PL/SQL block through a *i*SQL*Plus substitution variable. Test the PL/SQL block.
    d. Display the department that you updated.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Education		1700

Practice 3 (continued)

4. Create a PL/SQL block that deletes the department that you created in exercise 2. Save the PL/SQL block in a file named `p3q4.sql`. by clicking the `Save Script` button. Save the script with a `.sql` extension.
   a. Use the `DEFINE` command to provide the department ID.

      ```
      DEFINE p_deptno=280
      ```

   b. Pass the value to the PL/SQL block through a *i*SQL*Plus substitution variable. Print to the screen the number of rows affected.
   c. Test the PL/SQL block.

G_RESULT
1 row(s) deleted.

   d. Confirm that the department has been deleted.

      no rows selected