Using SQL Statements within a PL/SQL Block

Objectives

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

- Determine the SQL statements that can be directly included in a PL/SQL executable block
- Manipulate data with DML statements in PL/SQL
- Use transaction control statements in PL/SQL
- Make use of the `INTO` clause to hold the values returned by a SQL statement
- Differentiate between implicit cursors and explicit cursors
- Use SQL cursor attributes

In this lesson, you learn to embed standard SQL `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `MERGE` statements in PL/SQL blocks. You learn how to include data manipulation language (DML) and transaction control statements in PL/SQL. You learn the need for cursors and differentiate between the two types of cursors. The lesson also presents the various SQL cursor attributes that can be used with implicit cursors.

SQL Statements in PL/SQL

- Retrieve a row 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.

In a PL/SQL block, you use SQL statements to retrieve and modify data from the database table. PL/SQL supports data manipulation language (DML) and transaction control commands. You can use DML commands to modify the data in a database table. However, remember the following points while using DML statements and transaction control commands in PL/SQL blocks:

- The `END` keyword signals the end of a PL/SQL block, not the end of a transaction. Just as a block can span multiple transactions, a transaction can span multiple blocks.
- PL/SQL does not directly support data definition language (DDL) statements such as `CREATE TABLE`, `ALTER TABLE`, or `DROP TABLE`. PL/SQL supports early binding, which cannot happen if applications have to create database objects at run time by passing values. DDL statements cannot be directly executed. These statements are dynamic SQL statements. Dynamic SQL statements are built as character strings at run time and can contain placeholders for parameters. Therefore, you can use dynamic SQL to execute your DDL statements in PL/SQL. The details of working with dynamic SQL are covered in the course titled *Oracle Database: Develop PL/SQL Program Units*.
- PL/SQL does not directly support data control language (DCL) statements such as `GRANT` or `REVOKE`. You can use dynamic SQL to execute them.

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];
```


Use the SELECT statement to retrieve data from the database.

Guidelines for Retrieving Data in PL/SQL

- Terminate each SQL statement with a semicolon (;).
- Every value retrieved must be stored in a variable by using the INTO clause.
- The WHERE clause is optional and can be used to specify input variables, constants, literals, and PL/SQL expressions. However, when you use the INTO clause, you should fetch only one row; using the WHERE clause is required in such cases.

select_list	List of at least one column; can include SQL expressions, row functions, or group functions
variable_name	Scalar variable that holds the retrieved value
record_name	PL/SQL record that holds the retrieved values
table	Specifies the database table name
condition	Is composed of column names, expressions, constants, and comparison operators, including PL/SQL variables and constants

- Specify the same number of variables in the `INTO` clause as the number of database columns in the `SELECT` clause. Be sure that they correspond positionally and that their data types are compatible.
- Use group functions, such as `SUM`, in a SQL statement, because group functions apply to groups of rows in a table.

SELECT Statements in PL/SQL

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

```
DECLARE
 v_fname VARCHAR2(25);
BEGIN
 SELECT first_name INTO v_fname
 FROM employees WHERE employee_id=200;
 DBMS_OUTPUT.PUT_LINE(' First Name is : '||v_fname);
END;
/
```

```
anonymous block completed
 First Name is : Jennifer
```

ORACLE

INTO Clause

The INTO clause is mandatory and occurs between the SELECT and FROM clauses. It is used to specify the names of variables that hold the values that SQL returns from the SELECT clause. You must specify one variable for each item selected, and the order of the variables must correspond with the items selected.

Use the INTO clause to populate either PL/SQL variables or host variables.

Queries Must Return Only One Row

SELECT statements within a PL/SQL block fall into the ANSI classification of embedded SQL, for which the following rule applies: Queries must return only one row. A query that returns more than one row or no row generates an error.

PL/SQL manages these errors by raising standard exceptions, which you can handle in the exception section of the block with the NO_DATA_FOUND and TOO_MANY_ROWS exceptions. Include a WHERE condition in the SQL statement so that the statement returns a single row. You learn about exception handling in the lesson titled “Handling Exceptions.”

Note: In all cases where DBMS_OUTPUT.PUT_LINE is used in the code examples, the SET SERVEROUTPUT ON statement precedes the block.

How to Retrieve Multiple Rows from a Table and Operate on the Data

A `SELECT` statement with the `INTO` clause can retrieve only one row at a time. If your requirement is to retrieve multiple rows and operate on the data, you can make use of explicit cursors. You are introduced to cursors later in this lesson and learn about explicit cursors in the lesson titled “Using Explicit Cursors.”

Retrieving Data in PL/SQL: Example

Retrieve `hire_date` and `salary` for the specified employee.

```
DECLARE
 v_emp_hiredate   employees.hire_date%TYPE;
 v_emp_salary     employees.salary%TYPE;
BEGIN
  SELECT   hire_date, salary
  INTO     v_emp_hiredate, v_emp_salary
  FROM     employees
  WHERE    employee_id = 100;
  DBMS_OUTPUT.PUT_LINE ('Hire date is :'|| v_emp_hiredate);
  DBMS_OUTPUT.PUT_LINE ('Salary is :'|| v_emp_salary);
END;
/
```

```
anonymous block completed
Hire date is :17-JUN-03
Salary is :24000
```


In the example in the slide, the `v_emp_hiredate` and `v_emp_salary` variables are declared in the declarative section of the PL/SQL block. In the executable section, the values of the `hire_date` and `salary` columns for the employee with the `employee_id` `100` are retrieved from the `employees` table. Next, they are stored in the `v_emp_hiredate` and `v_emp_salary` variables, respectively. Observe how the `INTO` clause, along with the `SELECT` statement, retrieves the database column values and stores them in the PL/SQL variables.

Note: The `SELECT` statement retrieves `hire_date`, and then `salary`. The variables in the `INTO` clause must thus be in the same order. For example, if you exchange `v_emp_hiredate` and `v_emp_salary` in the statement in the slide, the statement results in an error.

Retrieving Data in PL/SQL

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

Example:

```
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 of salary is ' || v_sum_sal);
END;
```

```
anonymous block completed
The sum of salary is 28800
```

ORACLE

In the example in the slide, the `v_sum_sal` and `v_deptno` variables are declared in the declarative section of the PL/SQL block. In the executable section, the total salary for the employees in the department with `department_id` 60 is computed using the SQL aggregate function `SUM`. The calculated total salary is assigned to the `v_sum_sal` variable.

Note: Group functions cannot be used in PL/SQL syntax. They must be used in SQL statements within a PL/SQL block as shown in the example in the slide.

For instance, you *cannot* use group functions using the following syntax:

```
V_sum_sal := SUM(employees.salary);
```

Naming Ambiguities

```
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;
/
```

```
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
01422. 00000 -  "exact fetch returns more than requested number of rows"
*Cause:    The number specified in exact fetch is less than the rows returned.
*Action:   Rewrite the query or change number of rows requested
```

ORACLE

In potentially ambiguous SQL statements, the names of database columns take precedence over the names of local variables.

The example shown in the slide is defined as follows: Retrieve the hire date and today's date from the `employees` table for `employee_id` `176`. This example raises an unhandled run-time exception because, in the `WHERE` clause, the PL/SQL variable names are the same as the database column names in the `employees` table.

The following `DELETE` statement removes all employees from the `employees` table, where the last name is not null (not just "King"), because the Oracle Server assumes that both occurrences of `last_name` in the `WHERE` clause refer to the database column:

```
DECLARE
   last_name VARCHAR2(25) := 'King';
BEGIN
   DELETE FROM employees WHERE last_name = last_name;
. . .
```

Naming Conventions

- Use a naming convention to avoid ambiguity in the `WHERE` clause.
- Avoid using database column names as identifiers.
- 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.
- The names of variables take precedence over the function names.

ORACLE

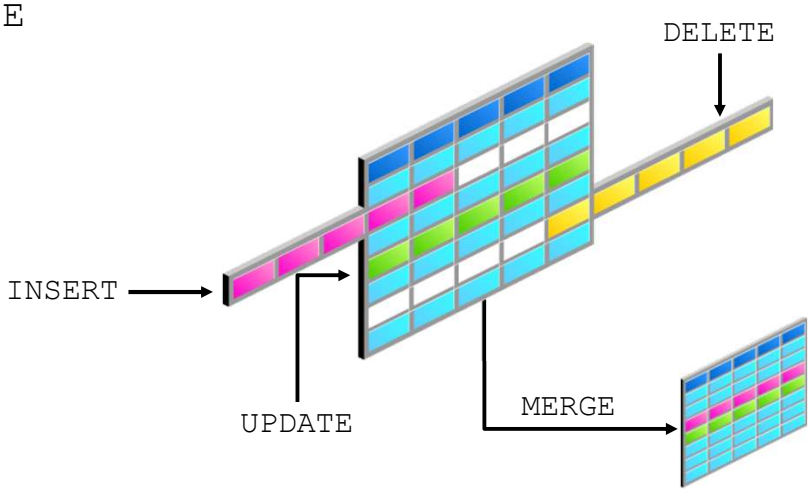
Avoid ambiguity in the `WHERE` clause by adhering to a naming convention that distinguishes database column names from PL/SQL variable names.

- 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.

Using PL/SQL to Manipulate Data

Make changes to database tables by using DML commands:

- INSERT
- UPDATE
- DELETE
- MERGE



ORACLE

You manipulate data in the database by using DML commands. You can issue DML commands such as INSERT, UPDATE, DELETE, and MERGE without restriction in PL/SQL. Row locks (and table locks) are released by including the COMMIT or ROLLBACK statements in the PL/SQL code.

- The INSERT statement adds new rows to the table.
- The UPDATE statement modifies existing rows in the table.
- The DELETE statement removes rows from the table.
- The MERGE statement selects rows from one table to update or insert into another table. The decision whether to update or insert into the target table is based on a condition in the ON clause.

Note: MERGE is a deterministic statement. That is, you cannot update the same row of the target table multiple times in the same MERGE statement. You must have INSERT and UPDATE object privileges on the target table and SELECT privilege on the source table.

Inserting Data: Example

Add new employee information to the EMPLOYEES table.

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

ORACLE

In the example in the slide, an INSERT statement is used within a PL/SQL block to insert a record into the employees table. While using the INSERT command in a PL/SQL block, you can:

- Use SQL functions such as USER and CURRENT_DATE
- Generate primary key values by using existing database sequences
- Derive values in the PL/SQL block

Note: The data in the employees table needs to remain unchanged. Even though the employees table is not read-only, inserting, updating, and deleting are not allowed on this table to ensure consistency of output. Therefore, the command rollback is used as shown in the code for slide 15_sa in code_ex_05.sql.

Updating Data: Example

Increase the salary of all employees who are stock clerks.

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

```
anonymous block completed
FIRST_NAME           SALARY
-------------------- ----------------------
Julia                4000
Irene                3500
James                3200
Steven               3000
. . .
Curtis               3900
Randall              3400
Peter                3300

20 rows selected
```


There may be ambiguity in the `SET` clause of the `UPDATE` statement because, although the identifier on the left of the assignment operator is always a database column, the identifier on the right can be either a database column or a PL/SQL variable. Recall that if column names and identifier names are identical in the `WHERE` clause, the Oracle Server looks to the database first for the name.

Remember that the `WHERE` clause is used to determine the rows that are affected. If no rows are modified, no error occurs (unlike the `SELECT` statement in PL/SQL).

Note: PL/SQL variable assignments always use `:=`, and SQL column assignments always use `=`.

Deleting Data: Example

Delete rows that belong to department 10 from the `employees` table.

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


The `DELETE` statement removes unwanted rows from a table. If the `WHERE` clause is not used, all the rows in a table can be removed if there are no integrity constraints.

Merging Rows

Insert or update rows in the `copy_emp` table to match the `employees` table.

```
BEGIN
MERGE INTO copy_emp c
     USING employees e
     ON (e.employee_id = c.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;
/
```


The `MERGE` statement inserts or updates rows in one table by using data from another table. Each row is inserted or updated in the target table depending on an equijoin condition.

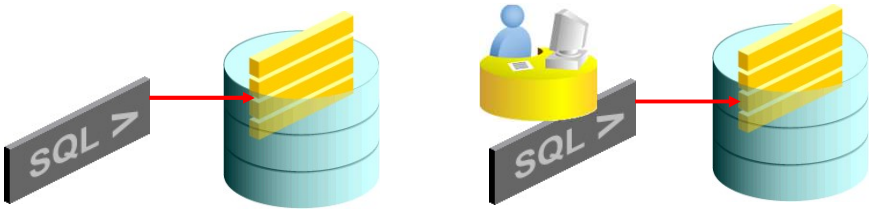
The example shown matches the `empno` column in the `copy_emp` table to the `employee_id` column in the `employees` table. If a match is found, the row is updated to match the row in the `employees` table. If the row is not found, it is inserted into the `copy_emp` table.

The complete example of using `MERGE` in a PL/SQL block is shown on the next slide.

```
BEGIN
MERGE INTO copy_emp c
     USING employees e
     ON (e.employee_id = c.empno)
   WHEN MATCHED THEN
     UPDATE SET
       c.first_name     = e.first_name,
       c.last_name      = e.last_name,
       c.email          = e.email,
       c.phone_number   = e.phone_number,
       c.hire_date      = e.hire_date,
       c.job_id         = e.job_id,
       c.salary         = e.salary,
       c.commission_pct = e.commission_pct,
       c.manager_id     = e.manager_id,
       c.department_id  = e.department_id
   WHEN NOT MATCHED THEN
     INSERT VALUES(e.employee_id,  e.first_name, e.last_name,
          e.email, e.phone_number, e.hire_date, e.job_id,
          e.salary, e.commission_pct, e.manager_id,
          e.department_id);
END;
/
```

SQL Cursor

- A cursor is a pointer to the private memory area allocated by the Oracle Server. It is used to handle the result set of a `SELECT` statement.
- There are two types of cursors: implicit and explicit.
  - **Implicit:** Created and managed internally by the Oracle Server to process SQL statements
  - **Explicit:** Declared explicitly by the programmer



Implicit cursor Explicit cursor

ORACLE

You have already learned that you can include SQL statements that return a single row in a PL/SQL block. The data retrieved by the SQL statement should be held in variables using the `INTO` clause.

Where Does the Oracle Server Process SQL Statements?

The Oracle Server allocates a private memory area called the *context area* for processing SQL statements. The SQL statement is parsed and processed in this area. The information required for processing and the information retrieved after processing are all stored in this area. You have no control over this area because it is internally managed by the Oracle Server.

A cursor is a pointer to the context area. However, this cursor is an implicit cursor and is automatically managed by the Oracle Server. When the executable block issues a SQL statement, PL/SQL creates an implicit cursor.

Types of Cursors

There are two types of cursors:

- **Implicit:** An *implicit cursor* is created and managed by the Oracle Server. You do not have access to it. The Oracle Server creates such a cursor when it has to execute a SQL statement.

- **Explicit:** As a programmer, you may want to retrieve multiple rows from a database table, have a pointer to each row that is retrieved, and work on the rows one at a time. In such cases, you can declare cursors explicitly depending on your business requirements. A cursor that is declared by programmers is called an *explicit cursor*. You declare such a cursor in the declarative section of a PL/SQL block.

SQL Cursor Attributes for Implicit Cursors

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

SQL%FOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement affected at least one row
SQL%NOTFOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement did not affect even one row
SQL%ROWCOUNT	An integer value that represents the number of rows affected by the most recent SQL statement

SQL cursor attributes enable you to evaluate what happened when an implicit cursor was last used. Use these attributes in PL/SQL statements but not in SQL statements.

You can test the SQL%ROWCOUNT, SQL%FOUND, and SQL%NOTFOUND attributes in the executable section of a block to gather information after the appropriate DML command executes. PL/SQL does not return an error if a DML statement does not affect rows in the underlying table. However, if a SELECT statement does not retrieve any rows, PL/SQL returns an exception.

Observe that the attributes are prefixed with SQL. These cursor attributes are used with implicit cursors that are automatically created by PL/SQL and for which you do not know the names. Therefore, you use SQL instead of the cursor name.

The SQL%NOTFOUND attribute is the opposite of SQL%FOUND. This attribute may be used as the exit condition in a loop. It is useful in UPDATE and DELETE statements when no rows are changed because exceptions are not returned in these cases.

You learn about explicit cursor attributes in the lesson titled “Using Explicit Cursors.”

SQL Cursor Attributes for Implicit Cursors

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

Example:

```
DECLARE
  v_rows_deleted VARCHAR2(30);
  v_empno employees.employee_id%TYPE := 176;
BEGIN
  DELETE FROM  employees
  WHERE employee_id = v_empno;
  v_rows_deleted := (SQL%ROWCOUNT ||
                        ' row deleted.');
  DBMS_OUTPUT.PUT_LINE (v_rows_deleted);

END;
```


The example in the slide deletes a row with `employee_id` `176` from the `employees` table. Using the `SQL%ROWCOUNT` attribute, you can print the number of rows deleted.

Quiz

When using the SELECT statement in PL/SQL, the INTO clause is required and queries can return one or more rows.

a. True
b. False

Answer: b

INTO Clause

The INTO clause is mandatory and occurs between the SELECT and FROM clauses. It is used to specify the names of variables that hold the values that SQL returns from the SELECT clause. You must specify one variable for each item selected, and the order of the variables must correspond with the items selected.

Use the INTO clause to populate either PL/SQL variables or host variables.

Queries Must Return Only One Row

SELECT statements within a PL/SQL block fall into the ANSI classification of embedded SQL, for which the following rule applies: Queries must return only one row. A query that returns more than one row or no row generates an error.

PL/SQL manages these errors by raising standard exceptions, which you can handle in the exception section of the block with the NO_DATA_FOUND and TOO_MANY_ROWS exceptions. Include a WHERE condition in the SQL statement so that the statement returns a single row. You learn about exception handling later in the course.

Summary

In this lesson, you should have learned how to:

- Embed DML statements, transaction control statements, and DDL statements in PL/SQL
- Use the `INTO` clause, which is mandatory for all `SELECT` statements in PL/SQL
- Differentiate between implicit cursors and explicit cursors
- Use SQL cursor attributes to determine the outcome of SQL statements

DML commands and transaction control statements can be used in PL/SQL programs without restriction. However, the DDL commands cannot be used directly.

A `SELECT` statement in a PL/SQL block can return only one row. It is mandatory to use the `INTO` clause to hold the values retrieved by the `SELECT` statement.

A cursor is a pointer to the memory area. There are two types of cursors. Implicit cursors are created and managed internally by the Oracle Server to execute SQL statements. You can use SQL cursor attributes with these cursors to determine the outcome of the SQL statement. Explicit cursors are declared by programmers.