Oracle PL/SQL - Advanced

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

1. Working with Procedures

What is a Procedure?

A **Procedure** is a **named PL/SQL block** stored in the database that performs one or more actions. It may or may not return values (unlike functions).

```
CREATE OR REPLACE  PROCEDURE procedure_name(   parameters )
IS
    -- variable delcation
BEGIN
    -- Code
END;
```

1. Working with Procedures

Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
    (parameter1 [IN | OUT | IN OUT] datatype, parameter2 [IN | OUT | IN OUT] datatype, ...)
IS
    -- Local variables
BEGIN
    -- Executable statements
EXCEPTION
    -- Exception handling
END procedure_name;
/
```

Example

```
CREATE OR REPLACE PROCEDURE raise_salary (
    p_emp_id    IN  employees.employee_id%TYPE,
    p_percent   IN  NUMBER
)
IS
BEGIN
    UPDATE employees
       SET salary = salary + (salary * p_percent / 100)
     WHERE employee_id = p_emp_id;
    COMMIT;
END raise_salary;
/
```

Executing a Procedure

```
EXEC raise_salary(101, 10);
```

Key Notes

- **IN**: Passes a value into the procedure (default).
- **OUT**: Returns a value to the caller.
- **IN OUT**: Passes a value in and may change it.

2. Working with Functions

What is a Function?

A **Function** is similar to a procedure but **must return a value** using the `RETURN` keyword.

Syntax

```
CREATE [OR REPLACE] FUNCTION function_name
    (parameter1 datatype, parameter2 datatype, ...)
RETURN return_datatype
IS
    -- Local variables
BEGIN
    -- Executable statements
    RETURN value;
EXCEPTION
```

Example

```
CREATE OR REPLACE FUNCTION get_annual_salary (
    p_emp_id IN employees.employee_id%TYPE
)
RETURN NUMBER
IS
    v_annual_salary NUMBER;
BEGIN
    SELECT salary * 12 INTO v_annual_salary
      FROM employees
     WHERE employee_id = p_emp_id;

    RETURN v_annual_salary;
END get_annual_salary;
/
```

Using a Function

```
-- From SQL:
SELECT get_annual_salary(101) FROM dual;

-- From PL/SQL block:
DECLARE
   v_salary NUMBER;
BEGIN
   v_salary := get_annual_salary(101);
   DBMS_OUTPUT.PUT_LINE('Annual Salary: ' || v_salary);
END;
/
```

3. Working with Packages

What is a Package?

A **Package** is a **collection of logically related PL/SQL objects** (procedures, functions, variables, cursors) stored together.

Packages have:

- **Specification (Header)** – Public elements.
- **Body** – Implementation.

Syntax

```
CREATE OR REPLACE PACKAGE package_name IS
   -- Public declarations
   PROCEDURE proc_name (...);
   FUNCTION func_name (...) RETURN datatype;
END package_name;
/

CREATE OR REPLACE PACKAGE BODY package_name IS
   PROCEDURE proc_name (...) IS
   BEGIN
      -- Implementation
   END proc_name;

   FUNCTION func_name (...) RETURN datatype IS
   BEGIN
      -- Implementation
   END func_name;
END package_name;
/
```

Example

```
CREATE OR REPLACE PACKAGE emp_pkg IS
   PROCEDURE raise_salary(p_emp_id IN NUMBER, p_percent IN NUMBER);
   FUNCTION get_salary(p_emp_id IN NUMBER) RETURN NUMBER;
END emp_pkg;
/

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
   PROCEDURE raise_salary(p_emp_id IN NUMBER, p_percent IN NUMBER) IS
   BEGIN
      UPDATE employees
         SET salary = salary + (salary * p_percent / 100)
       WHERE employee_id = p_emp_id;
      COMMIT;
   END raise_salary;

   FUNCTION get_salary(p_emp_id IN NUMBER) RETURN NUMBER IS
      v_salary NUMBER;
   BEGIN
      SELECT salary INTO v_salary FROM employees WHERE employee_id = p_emp_id;
      RETURN v_salary;
   END get_salary;
END emp_pkg;
/
```

Using a Package

```
EXEC emp_pkg.raise_salary(101, 10);
SELECT emp_pkg.get_salary(101) FROM dual;
```

4. Working with Triggers

What is a Trigger?

A **Trigger** is a stored PL/SQL block that **automatically executes** when a specified event occurs on a table or view.

Types

- **DML Triggers**: Fire on `INSERT`, `UPDATE`, `DELETE`
- **BEFORE / AFTER** triggers
- **ROW-level**

Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
   {BEFORE | AFTER} {INSERT | UPDATE | DELETE}
   ON table_name
   [FOR EACH ROW]
DECLARE
   -- Optional variable declarations
BEGIN
   -- Trigger logic
END;
/
```

Example

```
CREATE OR REPLACE TRIGGER trg_audit_emp
AFTER INSERT OR UPDATE OR DELETE ON employees
FOR EACH ROW
BEGIN
   INSERT INTO emp_audit_log (emp_id, action_date, action_type)
   VALUES (:OLD.employee_id, SYSDATE,
            CASE
               WHEN INSERTING THEN 'INSERT'
               WHEN UPDATING THEN 'UPDATE'
               WHEN DELETING THEN 'DELETE'
            END);
END;
/
```

5. Working with Views, Inline Views, and Correlated Subqueries

View

A **View** is a **stored SQL query** that presents data from one or more tables.

Syntax

```
CREATE [OR REPLACE] VIEW view_name AS
SELECT columns FROM table WHERE conditions;
```

Example

```
CREATE OR REPLACE VIEW emp_dept_view AS
SELECT e.employee_id, e.first_name, e.salary, d.department_name
  FROM employees e JOIN departments d USING(department_id);
```

Inline View

An **Inline View** is a **subquery in the FROM clause** of a query.

Example

```sql
SELECT department_id, avg_salary
FROM (
    SELECT department_id, AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id
)
WHERE avg_salary > 10000;
```

6. Working with Materialized Views

What is a Materialized View?

A **Materialized View** is a **precomputed table** that stores the results of a query physically, unlike a normal view which is virtual.

Used for performance optimization and data replication.

Syntax

```
CREATE MATERIALIZED VIEW mv_emp_summary
BUILD IMMEDIATE
REFRESH FAST ON COMMIT
AS
SELECT department_id, COUNT(*) AS emp_count, AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id;
```

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com