**What is a Procedure in Oracle SQL?**

a **procedure** in Oracle SQL is like a reusable block of code that contains a series of SQL commands and logic to perform a specific task. It's stored in the database, so you can call it whenever needed, without writing the same code over and over again.

**Key Features of Procedures:**

1. **Encapsulation**: Procedures allow you to encapsulate logic in a reusable and maintainable way.
2. **Parameterization**: You can pass parameters to procedures to influence their behaviour.
3. **Execution**: Procedures can be executed by other PL/SQL blocks, SQL queries, or through application code.
4. **Transaction control**: Procedures can handle transactions, meaning you can control commit and rollback actions.

**How to Create a Procedure in Oracle SQL?**

The basic syntax for creating a procedure in Oracle SQL is:

CREATE [OR REPLACE] PROCEDURE procedure\_name

**[ (**parameter1 **[**IN | OUT | IN OUT**]** datatype, parameter2 **[**IN | OUT | IN OUT**]** datatype, ...) **]**

IS

-- Declarations (optional)

BEGIN

-- SQL statements (procedure body)

-- Example: SELECT INTO, INSERT, UPDATE, DELETE statements

END procedure\_name;

**Breakdown:**

* **CREATE PROCEDURE**: This is the command to create the procedure.
* **procedure\_name**: Name of the procedure.
* **parameter1, parameter2**: Parameters that can be passed into the procedure. They can be of type IN (input), OUT (output), or IN OUT (both input and output).
* **IS**: Marks the start of the procedure body.
* **BEGIN**: The actual code and logic that defines the procedure.
* **END**: Marks the end of the procedure.

**Example 1: Simple Procedure without Parameters**

This example demonstrates a basic procedure that simply retrieves data from a table.

**Step 1: Creating a Procedure**

CREATE OR REPLACE PROCEDURE get\_all\_employees

IS

BEGIN

-- Fetching all employees

FOR emp IN (SELECT \* FROM employees)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || emp.first\_name || ' ' || emp.last\_name);

END LOOP;

END get\_all\_employees;

**Step 2: Calling the Procedure**

Once the procedure is created, you can call it using the EXEC command (or EXECUTE in SQL\*Plus).

EXEC get\_all\_employees;

**Example 2: Procedure with Parameters**

This example shows how to create a procedure that accepts parameters and performs an action based on them.

**Step 1: Creating the Procedure**

CREATE OR REPLACE PROCEDURE get\_employee\_by\_id(p\_emp\_id IN NUMBER)

IS

v\_first\_name employees.first\_name%TYPE;

v\_last\_name employees.last\_name%TYPE;

BEGIN

SELECT first\_name, last\_name

INTO v\_first\_name, v\_last\_name

FROM employees

WHERE employee\_id = p\_emp\_id;

DBMS\_OUTPUT.PUT\_LINE ('Employee Name: ' || v\_first\_name || ' ' || v\_last\_name);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No employee found with ID: ' || p\_emp\_id);

END get\_employee\_by\_id;

* **p\_emp\_id**: An IN parameter to accept an employee ID.
* **v\_first\_name, v\_last\_name**: Variables used to store the fetched employee details.
* The SELECT INTO statement is used to retrieve data from the employees table.
* The DBMS\_OUTPUT.PUT\_LINE procedure prints the results.

**Step 2: Calling the Procedure**

To call this procedure, you pass an employee ID as a parameter.

EXEC get\_employee\_by\_id(101);

**Example 3: Procedure with IN, OUT, and IN OUT Parameters**

This example demonstrates a procedure that has different types of parameters: IN, OUT, and IN OUT.

**Step 1: Creating the Procedure**

CREATE OR REPLACE PROCEDURE update\_employee\_salary(

p\_emp\_id IN NUMBER,

p\_new\_salary IN NUMBER,

p\_old\_salary OUT NUMBER

) IS

BEGIN

-- Fetch current salary

SELECT salary

INTO p\_old\_salary

FROM employees

WHERE employee\_id = p\_emp\_id;

-- Update salary

UPDATE employees

SET salary = p\_new\_salary

WHERE employee\_id = p\_emp\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Employee not found.');

END update\_employee\_salary;

* **IN parameter**: p\_emp\_id and p\_new\_salary are input parameters.
* **OUT parameter**: p\_old\_salary is used to return the previous salary of the employee.

**Step 2: Calling the Procedure**

To call the procedure and get the old salary:

DECLARE

v\_old\_salary NUMBER;

BEGIN

-- Calling the procedure

update\_employee\_salary(101, 75000, v\_old\_salary);

DBMS\_OUTPUT.PUT\_LINE('Old Salary: ' || v\_old\_salary);

END;

**Advantages of Using Procedures**

1. **Modularity**: Procedures allow complex operations to be broken into smaller, manageable pieces.
2. **Reusability**: Once written, you can call procedures from multiple places (SQL scripts, applications, etc.).
3. **Performance**: Stored procedures execute on the server, reducing network overhead.
4. **Security**: You can encapsulate sensitive logic within procedures and grant users access to execute the procedure, not the underlying data.

**How to Alter or Drop a Procedure**

1. **Alter**: If you want to change a procedure, use CREATE OR REPLACE PROCEDURE to modify it.

CREATE OR REPLACE PROCEDURE procedure\_name IS

-- New logic here

END procedure\_name;

1. **Drop**: If you want to remove a procedure from the database, use the DROP PROCEDURE command:

DROP PROCEDURE procedure\_name;

**Conclusion**

Procedures are an essential part of PL/SQL programming in Oracle. They provide a way to encapsulate reusable logic, make SQL scripts more modular, and improve maintainability. You can use them to interact with the database, manipulate data, and perform business logic in a clean and efficient manner.