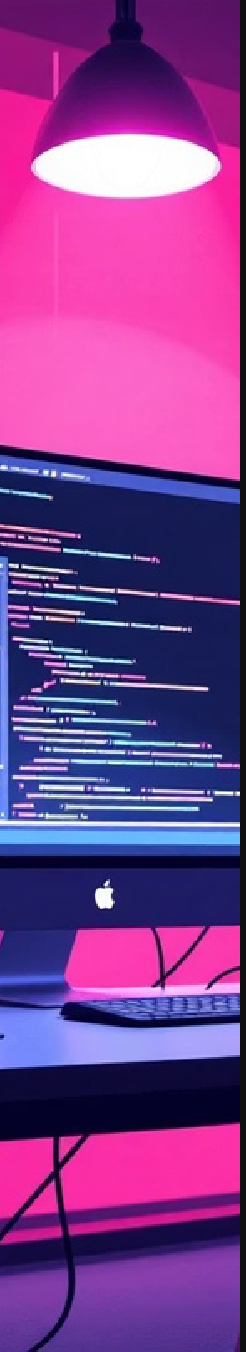


Oracle PL/SQL Stored Procedures

A**stored procedure** is a named PL/SQL blockstored permanently in the Oracledatabase. It groups SQLstatements and business logic into a reusable unit, eliminatingthe needtorewrite code repeatedly.

Think of it as a database function that can accept parameters, perform operations, and be called whenever needed.

Stored Procedure Syntax & Structure

01

Declaration

Use CREATEORREPLACE PROCEDURE to define the procedure name and parameters

02

Parameters

DefineIN(input),OUT (output), or IN OUT parameters with their data types

03

Body

WriteSQL statements and PL/SQL logic between BEGIN and END keywords

04

Compilation

Executewithforward slash (/) to compile and store in the database

```
CREATE OR REPLACE PROCEDURE procedure_name (
param1 IN datatype,
param2 OUT datatype
) AS
BEGIN
 -- SQL or PL/SQL statements
 NULL;
END;
/
```

Complete Example: Employee Management

Creating the Procedure

```
CREATE OR REPLACE PROCEDURE add_employee (
p_emp_id IN NUMBER,
p_name IN VARCHAR2,
p_salary IN NUMBER
) AS
BEGIN
 INSERT INTO employees
(emp_id, name, salary)
VALUES (p_emp_id, p_name, p_salary);

   COMMIT;
END;
/
```

Executing the Procedure

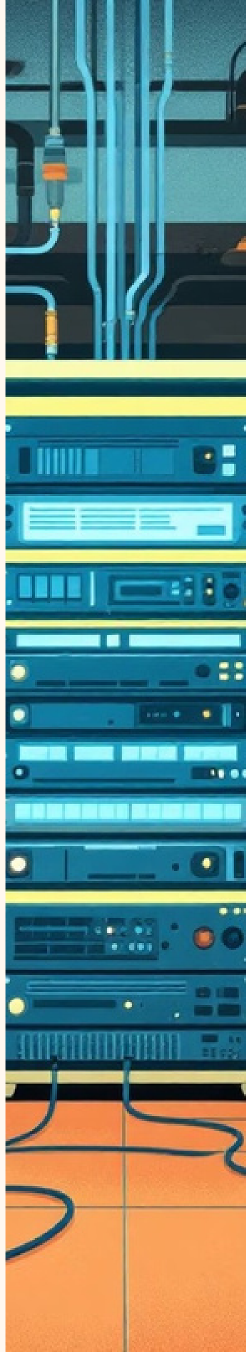
```
-- Method 1: Using EXECUTE
EXECUTE add_employee(101, 'John', 50000);

-- Method 2: In PL/SQL block
BEGIN
 add_employee(102, 'Alice', 60000);
 add_employee(103, 'Bob', 55000);
END;
/
```

Key Benefits: Stored procedures improve performance through pre-compilation, enhance security by reducing SQL injection risks, and promote code reusability across applications.

Understanding Database Triggers

Atriggeris a specialPL/SQL blockstoredinthe database thatexecutesautomaticallywhen specific events occur on a table.Unlike proceduresor functions, triggers fire without manualexecutionwhenever INSERT, UPDATE, or DELETE operations happen.

Triggers serve as database guardians, monitoring table changes and responding with predefined actions like auditing, validation, or data synchronization.



Trigger Syntax Breakdown

1. **CREATE OR REPLACE TRIGGER**

   Definesanewtriggerorreplacesexisting one with the same name

2. **trigger_name**

   Uniqueidentifierfollowing Oracle naming conventions

3. **AFTER INSERT OR UPDATE OR DELETE**

   Specifies timing (BEFORE/AFTER) and triggering events

4. **ON table_name**

   Targettablewhereevents are monitored

5. **FOR EACH ROW**

   Makes itarow-leveltrigger (fires once per affected row)

```
CREATE OR REPLACE TRIGGER trigger_name
AFTER INSERT OR UPDATE OR DELETE ON table_name
FOR EACH ROW
BEGIN
 -- Trigger action
 NULL;
END;
/
```

Practical Example: Employee Audit Trigger

Trigger Code Analysis

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

Key Components

- :OLD.emp_id - References original column value
- SYSDATE - Captures current timestamp
- CASE statement - Determines operation type
- INSERTING/UPDATING/DELETING - Built-in predicates

Automatic Execution: When you run INSERT INTO employees (emp_id, name, salary) VALUES (104, 'Eve', 80000); the trigger fires automatically and logs the action to emp_audit table.