East West University
Department of Computer Science and Engineering
Course: CSE464
Instructor: Antu Chowdhury

Lab 04: Working with PL/SQL Procedures, Functions, and Triggers

Lab Objectives:

- Understand how to **create and use PL/SQL procedures** to perform specific tasks such as DML operations (INSERT, UPDATE, DELETE).
- Learn the difference between **procedures and functions**, especially focusing on return values and parameter modes (IN, OUT, IN OUT).
- Practice **using OUT parameters** to return values from procedures.
- Execute procedures using both **EXEC** and **anonymous PL/SQL blocks**.
- Learn the syntax and use of **PL/SQL functions** that return calculated results.
- Understand **triggers** in PL/SQL, their firing events (BEFORE/AFTER INSERT, UPDATE, DELETE), and how to use bind variables (:NEW, :OLD) in triggers.
- Write simple triggers for **automatic backup of data** during DELETE or INSERT operations.

Topics Covered:

- Creating and running **PL/SQL procedures** with IN and OUT parameters
- Understanding when to use **procedures vs functions**
- Writing PL/SQL functions that **return calculated values** (e.g., bonus calculation)
- Executing procedures with `EXEC` and anonymous blocks
- Syntax and examples of **PL/SQL triggers**
- Using **bind variables** `:NEW` and `:OLD` inside triggers
- Creating **row-level triggers** to log changes automatically

Lab Outcomes:

By the end of this lab session, students were able to:

1. Distinguish between procedures and functions and create functions that return values using the `RETURN` statement.
2. Execute procedures using both `EXEC` and anonymous PL/SQL blocks.
3. Create **PL/SQL triggers** that automatically perform tasks such as backing up deleted or inserted data.
4. Use bind variables `:NEW` and `:OLD` in triggers to access row values before and after DML operations.

PL/SQL Procedure

A procedure in PL/SQL is a named block of code designed to perform one or more specific tasks. Procedures are commonly used to execute DML (Data Manipulation Language) operations such as INSERT, UPDATE, or DELETE on the database.

- Unlike functions, a procedure does not require a return value.
- In PL/SQL, both IS and AS can be used to begin the declaration section of a procedure—they are functionally equivalent.
- Procedures can accept input (IN), output (OUT), or both input and output (IN OUT) parameters.

Key Benefits of Using Procedures

- ✅ Code Reusability: Write once, use many times—procedures can be called from different parts of an application.
- ✅ Security: Users can be granted permission to execute procedures without direct access to underlying tables, ensuring better control and protection of sensitive data.

Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
(
   parameter_name [IN | OUT | IN OUT] datatype,
   ...
)
IS | AS
BEGIN
   -- Procedure body (statements go here)
END procedure_name;
```

✅ 1. See the List of Existing Procedures

To view all stored procedures owned by the current user:

```
SELECT object_name, status
FROM user_objects
WHERE object_type = 'PROCEDURE';
```

✅ How to Run a Procedure

1. Using EXEC (shortcut command):

```
SQL> EXEC procedure_name;
```

- This is a quick way to execute a procedure.

2. Using an anonymous PL/SQL block:

```
SQL> BEGIN
  2    procedure_name;
  3  END;
  4  /
```

- This method is more flexible and can handle multiple statements or exception handling.

Example 1: Write a PL/SQL procedure named `welcome` that prints the message "WELCOME TO PROCEDURE".

```
CREATE OR REPLACE PROCEDURE welcome
IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('WELCOME TO PROCEDURE');
END;
```

Example 2: Write a PL/SQL procedure named `INSERT_EMPLOYEE` that inserts a new employee's ID, name, and salary into the EMPLOYEE table.

```
CREATE OR REPLACE PROCEDURE INSERT_EMPLOYEE
(E_ID IN NUMBER, NAME IN VARCHAR2, SALARY IN NUMBER)
IS
BEGIN
    INSERT INTO EMPLOYEE VALUES(E_ID, NAME, SALARY);
END INSERT_EMPLOYEE;
```

Example 3: Write a PL/SQL procedure `update_emp` that increases the salary of an employee and returns the updated salary using an OUT parameter.

```
CREATE OR REPLACE PROCEDURE update_emp(
    id     IN NUMBER,
    amount IN NUMBER,
    d      OUT NUMBER
)
IS
BEGIN
    UPDATE employee
    SET salary = salary + amount
    WHERE e_id = id;

    SELECT salary INTO d
    FROM employee
    WHERE e_id = id;

    DBMS_OUTPUT.PUT_LINE('Updated Salary: ' || d);
END update_emp;
```

To execute:

```
VARIABLE new_salary NUMBER;
EXEC update_emp(5, 500, :new_salary);
```

Or,

```
DECLARE
    new_salary NUMBER;
BEGIN
    update_emp(3, 500, new_salary);
END;
```

PL/SQL Function

A PL/SQL Function is similar to a Procedure but must return a value. Functions are typically used to perform calculations and return results.

- Must include a RETURN type (mandatory).
- Use the RETURN statement to provide the value.
- Accepts IN, OUT, or IN OUT parameters (but usually just IN).

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name
(
    parameter_name [IN | OUT | IN OUT] datatype,
    ...
)
RETURN return_datatype
IS|AS
    -- variable declarations
BEGIN
    -- function logic
    RETURN value;  -- mandatory
END function_name;
```

Example 1: Write a PL/SQL function `get_bonus` that takes an employee's salary and returns 10% of it as a bonus.

```
CREATE OR REPLACE FUNCTION get_bonus (
    salary IN NUMBER
)
RETURN NUMBER
IS
    bonus NUMBER;
BEGIN
    bonus := salary * 0.10;
    RETURN bonus;
END get_bonus;
```

✅ To Call the Function:

```
-- In PL/SQL block:
DECLARE
    result NUMBER;
BEGIN
    result := get_bonus(5000);
    DBMS_OUTPUT.PUT_LINE('Bonus: ' || result);
END;
```

Trigger in PL/SQL:

A trigger is a named PL/SQL block that is executed automatically by the database in response to a specified DML event (such as INSERT, UPDATE, or DELETE) on a table or view.

Scenario:

To automatically insert data into a 'Backup' table whenever data is deleted from a 'Main' table in PL/SQL, you can create a trigger on the 'Main' table. This trigger will fire whenever data is deleted from ‘Main table’ and insert the old row's data into the 'Backup' table.

Syntax of a Trigger:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER }
{INSERT | UPDATE | DELETE}
ON table_name
[FOR EACH ROW]
BEGIN
   -- PL/SQL statements (trigger body)
END;
```

Part	Description
BEFORE / AFTER	When the trigger should fire (before/after the DML event)
INSERT / UPDATE / DELETE	Event that causes the trigger to fire
FOR EACH ROW	Trigger fires once per row (Row-level trigger).
BEGIN ... END;	The body (logic) of the trigger

Bind Variables in Triggers

Bind Variable	Description
:NEW.column_name	Refers to the new value being inserted/updated
:OLD.column_name	Refers to the existing value before update/delete

Pre-Requiste_Table Structure: `BACKUP_EMP`

```
CREATE TABLE BACKUP_EMP (
  E_ID        NUMBER PRIMARY KEY,
  NAME        VARCHAR2(50),
  SALARY      NUMBER)
```

Example 1: Create a trigger `T1` on the EMPLOYEE table that takes a backup of the deleted row into the BACKUP_EMP table before a DELETE operation.

```
CREATE OR REPLACE TRIGGER T1
BEFORE DELETE ON EMPLOYEE
FOR EACH ROW
BEGIN
    INSERT INTO BACKUP_EMP VALUES(:OLD.E_ID,:OLD.NAME,:OLD.SALARY);
END;
```

Example 2: Create a trigger `t2` that inserts a row into the BACKUP_EMP table after a new row is inserted into the EMPLOYEE table.

```
CREATE OR REPLACE TRIGGER t2
AFTER INSERT ON EMPLOYEE
FOR EACH ROW
BEGIN
    INSERT INTO BACKUP_EMP VALUES (:NEW.E_ID, :NEW.NAME, :NEW.SALARY);
END;
```

✅ IF trigger is being created multiple times:

Run this query to check **how many triggers** are on the `EMPLOYEE` table:

```
SELECT trigger_name
FROM user_triggers
WHERE table_name = 'EMPLOYEE';
```

If you see **more than one trigger**, then that explains the duplicate rows.

□ How to fix it:

1. Drop extra triggers:

```
DROP TRIGGER t1;
DROP TRIGGER t5;
```

Let's now implement a **trigger** on the `EMPLOYEE` table that:

1. Logs the **delete time** using `SYSDATE`
2. Tracks the **username** of the person who performed the deletion using `USER`

✅ Step-by-step Instructions

◆ Step 1: Modify your backup table

If your table `BACKUP_EMP` doesn't already have these columns, add them:

```
ALTER TABLE BACKUP_EMP ADD (
  DELETED_AT DATE,
  DELETED_BY VARCHAR2(50)
);
```

◆ Step 2: Create or Replace the Trigger

```
CREATE OR REPLACE TRIGGER t3
BEFORE DELETE ON EMPLOYEE
FOR EACH ROW
BEGIN
  INSERT INTO BACKUP_EMP
  VALUES (:OLD.E_ID, :OLD.NAME, :OLD.SALARY, SYSDATE, USER);
END;
/
```

Problem Practice

Assume you have a table named `students` with the following attributes:

- student_id
- name
- semester_fee
- current_semester
- dept_name

---

1. **Write a PL/SQL procedure** named `insert_student` that inserts a new student record into the `students` table using **IN** parameters for all columns.

---

2. **Write a PL/SQL procedure** named `update_fee` that updates the `semester_fee` of a student by a given amount. The procedure should use an **OUT** parameter to return the updated fee.

---

3. **Write a PL/SQL function** named `calculate_scholarship` that takes a semester fee as input and returns 15% of it as the scholarship amount.

---

4. **Create a table** named `backup_students` with the same structure as the `students` table, but include two additional columns:

- `deleted_at` of type DATE
- `deleted_by` of type VARCHAR2(50)

---

5. **Write a row-level BEFORE DELETE trigger** on the `students` table that inserts the deleted row into the `backup_students` table along with the current date (`SYSDATE`) and the username of the person who performed the delete (`USER`).

---

6. **Create a trigger** named `backup_student_before_update` on the `students` table that takes a backup of the old row into the `backup_students` table **before** an UPDATE operation.

7. **Create a trigger** named `backup_student_after_insert` on the `students` table that inserts the newly inserted row into the `backup_students` table **after** an INSERT operation.

Submission Instructions:

1. For each question:
    - Run the PL/SQL code in your database environment.
    - Take screenshots of your SQL code and its output.
    - Paste the code and screenshots into a Word document (.doc or .docx).
2. Save the document as a PDF named:
   `<YourStudentID>_LAB04.pdf`
   (Replace `<YourStudentID>` with your actual student ID.)
3. Submit the PDF file using the assignment section in your classroom.