Database Systems Lab

- ❖ Maintain discipline during the lab.
- ❖ Listen and follow the instructions as they are given.
- ❖ Just raise hand if you have any problem.

MAIN OBJECTIVES:

Procedures in pl/sql
- Mode of parameters
- Example for out parameter
- Example for in parameter
- Example for in/out parameter

Function in pl/sql
- Syntax
- Example

Triggers in pl/sql
- Elements in a triggers
- Component of triggers
- Triggers body
- Syntax
- example

PROCEDURES IN PL/SQL:

Procedures are written for doing specific tasks. The general syntax of procedure is

SYNTAX:

CREATE OR REPLACE PROCEDURE <Pro_Name> (Par_Name1 [IN/OUT/ IN OUT] Par_Type1, ….) IS (Or we can write AS)
Local declarations;
BEGIN
PL/SQL Executable statements;
……. EXCEPTION
Exception Handlers;
END <Pro_Name>;

MODE OF PARAMETERS

1) **IN MODE :-** IN mode is used to pass a value to Procedure/Function. Inside the procedure/function, IN acts as a constant and any attempt to change its value causes compilation error.
2) **OUT MODE :** The OUT parameter is used to return value to the calling routine. Any attempt to refer to the value of this parameter results in null value.
3) **IN OUT MODE :** IN OUT parameter is used to pass a value to a subprogram and for getting the updated value from the subprogram.

1) SIMPLE PROGRAM TO ILLUSTRATE PROCEDURE.

-- Assume file name P1

EXAMPLE:

```
CREATE OR REPLACE PROCEDURE P1(A NUMBER) AS
BEGIN
DBMS_OUTPUT.PUT_LINE('A:'||A);
END P1;
/
```

Now write PL/SQL code to use procedure in separate file.

```
-- Assume file name testP1
DECLARE
BEGIN
P1(100);
END;
```

OUTPUT

```
SQL> @P1
Procedure created.
SQL>@testP1
A:100
PL/SQL procedure successfully completed.
```

2) PROGRAM TO ILLUSTRATE PROCEDURE WITH IN MODE PARAMETER.

EXAMPLE:

```
-- Assume file name P2
CREATE OR REPLACE PROCEDURE P2(A IN NUMBER) AS
BEGIN
DBMS_OUTPUT.PUT_LINE('A:'||A);
END P2;
/
-- Assume file name testP2
DECLARE
X NUMBER;
BEGIN
X:=10;
DBMS_OUTPUT.PUT_LINE('X:'||X);
P2(X);
DBMS_OUTPUT.PUT_LINE('X:'||X);
END;
/
```

OUTPUT

```
SQL> @P2
Procedure created.
SQL>@testP2
X:10
A:10
```

```
X:10
PL/SQL procedure successfully completed.
```

3) Program to illustrate Procedure with OUT mode parameter.

```
-- Assume file name P3
CREATE OR REPLACE PROCEDURE P3(A OUT NUMBER) AS
 BEGIN
 A:=100;
 DBMS_OUTPUT.PUT_LINE('A:'|| A);
 END P3;
 /

-- Assume file name testP3
DECLARE
X NUMBER;
BEGIN
X:=50;
DBMS_OUTPUT.PUT_LINE('X:'||X);
P3(X);
DBMS_OUTPUT.PUT_LINE('X:'||X);
END;
/
```

Output

```
SQL> @P3
Procedure created.
SQL>@testP3
X:50
A:100
X:100
PL/SQL procedure successfully completed.
```

4) Program to illustrate Procedure with OUT mode parameter.

```
-- Assume file name P4
CREATE OR REPLACE PROCEDURE P4(A OUT NUMBER) AS
BEGIN
DBMS_OUTPUT.PUT_LINE('A:'||A);
END P4;
/

-- Assume file name testP4
DECLARE
X NUMBER;
BEGIN
X:=10;
DBMS_OUTPUT.PUT_LINE('X:'||X);
```

```
P4(X);
DBMS_OUTPUT.PUT_LINE('X:'||X);
END;
/
```

Output

```
SQL> @P4
Procedure created.
SQL>@testP4
X:10
A:
X:
PL/SQL procedure successfully completed.
```

5) Program to illustrate Procedure with IN OUT mode parameter.

```
--Assume file name P5
CREATE OR REPLACE PROCEDURE P5(A IN OUT NUMBER) AS
BEGIN
DBMS_OUTPUT.PUT_LINE('A:' || A);
END P5;
/

-- Assume file name testP5
DECLARE
X NUMBER;
BEGIN
X:=10;
DBMS_OUTPUT.PUT_LINE('X:'|| X);
P5(X);
DBMS_OUTPUT.PUT_LINE('X:'|| X);
END;
/
```

Output

```
SQL> @P5
Procedure created.
SQL>@testP5
X:10
A:10
X:10
PL/SQL procedure successfully completed.
```

FUNCTIONS IN PL/SQL:

Similar to Procedure we can create Functions which can return one value to the calling program. The syntax of function is

SYNTAX:

```
CREATE OR REPLACE FUNCTION <Fun_Name> (Par_Name1 [IN/OUT/ IN OUT] Par_Type1, ….)
RETURN return_datatype IS
Local declarations;
BEGIN
PL/SQL Executable statements;
……. EXCEPTION
Exception Handlers;
END <Fun_Name>;
```

HOW TO WRITE FUNCTIONS?

- For writing Function we can directly type at SQL prompt or create a file.
  SQL> ed File_Name
- Type & save Function.
- To create Function (before calling from other program.)

SQL> @File_Name

- To use/call Function we have two ways.
  1) Write a PL/SQL code and include call in the code using
     x=Fun_Name(Par_List);
  2) You can execute from SQL Prompt as a select query
     SQL>Select Fun_Name(Par_List) from Dual;

Ex :- Program to illustrate Function. (Finding Square of a number)

EXAMPLE

```
CREATE OR REPLACE FUNCTION FUN(A NUMBER)
RETURN NUMBER IS
BEGIN
RETURN (A*A);
END FUN;
/

-- Assume file name testFun
DECLARE
X NUMBER:=&X;
S NUMBER;
BEGIN
S:=FUN(X);
DBMS_OUTPUT.PUT_LINE('SQUARE OF A NUMBER'|| S);
END;
```

/ OUTPUT

```
SQL> @Fun
```

```
Function created. SQL> @testFun
ENTER VALUE FOR X: 10
OLD  2: X NUMBER:=&X; NEW  2: X NUMBER:=10; SQUARE OF A NUMBER100
PL/SQL procedure successfully completed.
```

TRIGGER:

A trigger is a PL/SQL block or a PL/SQL procedure that executes implicitly whenever a particular event takes place. It can either be:

1. Application trigger: Fires whenever an event occurs with a particular application.
2. Database Trigger: Fires whenever a data event (such as DML) occurs on a schema or database.

ELEMENTS IN A TRIGGER:

· Trigger timing
- For table: BEFORE, AFTER
- For view: INSTEAD OF

· Trigger event: INSERT, UPDATE, OR DELETE
· Table name: On table, view
· Trigger Type: Row or statement
· When clause: Restricting condition
· Trigger body: PL/SQL block

FOR YOUR INFORMATION

“Before triggers” execute the trigger body before the triggering DML event on a table. These are frequently used to determine whether that triggering statement should be allowed to complete. This situation enables you to eliminate unnecessary processing of the
triggering statement and it eventual rollback in cases where an exception is raised in the triggering action.

“After triggers” are used when the triggering statement is to be completed before the triggering action and to perform a different action on the same triggering statement if a BEFORE trigger is already present.

“Instead of Triggers” are used to provide a transparent way of modifying views that cannot
be modified directly through SQL DML statements because the view is not inherently modifiable. You can write INSERT, UPDATE, and DELETE statements against the view. The
INSTEAD OF trigger works invisibly in the background performing the action coded in the
trigger body directly on the underlying tables.
.

.

Triggering user events:
o INSERT
o UPDATE
o DELETE

TRIGGER COMPONENTS:

o Statement: The trigger body executes once for the triggering event. This is the default. A statement trigger fires once, even if no rows are affected at all.
o Row: The trigger body executes once for each row affected by the triggering event. A row trigger is not executed if the triggering event affects no rows.

TRIGGER BODY:

The trigger body is a PL/SQL block or a call to a procedure.

SYNTAX:

```
CREATE [OR REPLACE] TRIGGER trigger_name
Timing
Event1 [OR event2 OR event3]
ON table_name
Trigger_body
```

EXAMPLE:

```
CREATE [OR REPLACE] TRIGGER secure_emp
Before insert on emp
BEGIN
IF(to_char(sysdate,’dy’) IN (‘SAT’,’SUN’)) OR
To_char(sysdate,’HH24’) NOT BETWEEN ‘08’ AND ‘18’)
THEN RAISE_APPLICATION_ERROR(-20500,
‘you may only insert into EMP during normal hours.’);
END IF;
END;
```

SYNTAX:

```
CREATE [OR REPLACE] TRIGGER trigger_name
Timing
Event1 [OR event2 OR event3]
ON table_name
Trigger_body
```

EXAMPLE:

```
 CREATE [OR REPLACE] TRIGGER secure_emp
Before insert on emp
BEGIN
IF(to_char(sysdate,'dy') IN ('SAT','SUN')) OR
To_char(sysdate,'HH24') NOT BETWEEN '08' AND '18')
THEN RAISE_APPLICATION_ERROR(-20500,
'you may only insert into EMP during normal hours.');
END IF;
END;
```

ACTIVITIES:

TASK 1:

A number of business rules apply to the EMP and DEPT tables. Some of which are as follows:-
i. Sales persons should always receive commission. Employees who are not sales persons should never receive a commission.
ii. The EMP table should contain exactly one PRESIDENT. Test your answer.
iii. An employee should never be manager of more than five employees. Test your answer.
iv. Salaries may only be increased, never decreased. Test your answer.
v. If a department moves to another location, each employee of that department automatically receives a salary raise of 2%.
Now implement the above business rules with the help of database triggers.

TASK 2:

Create a stored procedure that has two arguments *TID* and *duration*. The first argument takes training code for a training program from the calling environment and the second argument should return the duration of training in weeks to the calling environment. Then print the duration in the calling environment

TASK 3:

Suppose employee performance in projects is quantified as Excellent=4, Good=3, Fair=2, Bad=1, Poor=0 (referred to lab session 6). Write a stored function to pass the employee number and get the employee *total-grade* on project performance. Consider the schema in lab session 6. Write down a stored function that takes employee number and returns the percentage of projects in which employee performance is *excellent*.