SHREE ADARSH BCA COLLEGE – BOTAD
BCA SEM-6
SUB: Database Programming (PL/SQL) Using Oracle
UNIT –2: Advance PL/SQL Programming

❖ Introducing to PL/SQL Exception

- ✓ In PL/SQL, any kind of errors is treated as exceptions.
- ✓ An exception is defined as a special condition that changes the program execution flow.
- ✓ The PL/SQL provides you with a flexible and powerful way to handle such exceptions.
- ✓ PL/SQL catches and handles exceptions by using exception handler architecture.
- ✓ Whenever an exception occurs, it is raised. The current PL/SQL block execution halts and control is passed to a separate section called *exception section*.
- ✓ In the exception section, you can check what kind of exception has been occurred and handle it appropriately. This exception handler architecture enables separating the business logic and exception handling code hence make the program easier to read and maintain.
- ✓ **There are three types of exceptions:**
  - **1)** Predefined Exception / System Exception
  - **2)** Undefined Exception
  - **3)** User defined Exception

1) Predefined Exception / System Exception :

System exceptions are automatically raised by oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in oracle which is known as Named System Exceptions.

For example, NO_DATA_FOUND exception is raised if you select a non-existing record from the database.

Exception Name	Error No	Description
NO_DATA_FOUND	ORA-01403	It is raised when a SELECT INTO statement returns no rows.
INVALID_CURSOR	ORA-01001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
CURSOR_ALREADY_OPEN	ORA-65111	It is raised when you try to open an already open cursor.
TOO_MANY_ROWS	ORA-01422	It is raised when a SELECT INTO statement returns multiple rows.
ZERO_DIVIDE	ORA-01476	It is raised when number is divided by zero.

For example: It will handle ZERO_DIVIDE exception and we can write a code to handle the exception as given below.

```
DECLARE
        N1 number(3);
        N2 number(3);
```

```
        ans number(8,2);
BEGIN
        N1:=&no1;
        N2:=&no2;
        ans:= N1 / N2;
        dbms_output.put_line('Ans =  ' || ans);
EXCEPTION
        WHEN zero_divide THEN
        dbms_output.put.line('Divide Zero Error');
END;
/
```

2) Undefined Exception :

It's an oracle error that doesn't have a predefined exception (an undefined exception, sometimes called an "unnamed oracle error"). In this case you need to declare an exception and associate an oracle number with it. Those system exception for which oracle does not provide a name is known as unnamed system exception. These exceptions do not occur frequently.

Example:

```
DECLARE
        Error_ex EXCEPTION;
        PRAGMA EXCEPTION_INIT(Error_ex ,-2245);
BEGIN
        DELETE From employee WHERE eid=4;
EXCEPTION
        WHEN Error_ex THEN
        Dbms_output.put_line('Exception generated !');
END;
```

3) User defined Exception :

The user defined exception sometimes called programmer-defined exception is defined by you in a specific application.

- ✓ A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure DBMS_STANDARD. RAISE _ APPLICATION _ERROR.
- ✓ You can map exception names with specific Oracle errors using the EXCEPTION_INIT pragma. You can also assign a number and description to the exception using RAISE_APPLICATION_ERROR.
- ✓ **Example:**

```
DECLARE
        myex EXCEPTION;
        i NUMBER;
BEGIN
        FOR i IN (SELECT * FROM enum)
          LOOP
                IF i.eno = 3 THEN
```

```
                    RAISE myex;
                END IF;
            END LOOP;
        EXCEPTION
            WHEN myex THEN
            dbms_output.put.line('Employee number already exist in enum table.');
        END;
    /
```

❖ SUBPROGRAM:

A subprogram is an individual part of PL/SQL block which can be used to perform a specific task. There are two types of subprograms.

1. Procedures: These subprograms do not return a value directly; mainly used to perform an action.

2. Functions: These subprograms return a single value; mainly used to compute and return a value.

1. PL/SQL Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

Header: The header contains the name of the procedure and the parameters or variables passed to the procedure.

Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

- **Syntax for creating procedure:**

```
CREATE[OR REPLACE] PROCEDURE procedure_name [(parameter_name[IN |
OUT ] type [, ...])]
IS
   [declaration_section]
BEGIN
   executable_section
[EXCEPTION
   exception_section]
END ;
```

Here:

- ✓ **procedure_name:** specifies the name of the procedure.
- ✓ **[OR REPLACE]** option allows replaces an existing procedure.
- ✓ The **optional parameter list** contains name, mode and types of the parameters.
- ✓ **IS** marks the beginning of the body of the procedure and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section.
- ✓ **IN** indicates parameter value passed to the procedure. It is read only value.

- ✓ **OUT it indicates parameter value returned to the user.**

- **Create procedure example**
  In this example, we are going to insert record in user table. So you need to create user table first.

  Table creation:

- ✓ `create table user(id number(10) primary key,name varchar2(100));`

Now write the procedure code to insert record in user table.
Procedure Code:

```
SQL>edit pro1
create or replace procedure  INSERTUSER (id IN NUMBER, name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
/
```

- **How to execute a Stored Procedure?**
  There are two ways to execute a procedure.
  1) From the SQL prompt.
  EXECUTE [or EXEC] procedure_name;
  2) Within another procedure – simply use the procedure name.
  procedure_name;
- **Execute PROCEDURE**
  After write the PL/SQL Procedure you need to execute the procedure.
  SQL>@pro1
  Procedure created.
  PL/SQL procedure successfully completed.
- **PL/SQL program to call procedure**
  Let's see the code to call above created procedure.

```
BEGIN
  insertuser(101,'Rahul');
  dbms_output.put_line('record inserted successfully');
END;
/
```

- **PL/SQL Drop Procedure**
  - **Syntax for drop procedure**
    DROP PROCEDURE procedure_name;
  - **Example of drop procedure**
    DROP PROCEDURE pro1;

2. **PL/SQL Function**
   The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on

the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.
The function contains a header and a body.
Header: The header contains the name of the function and the parameters or variables passed to the function.
Body: The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

- **Syntax to create a function:**

```
CREATE [OR REPLACE] FUNCTION function_name[(parameter_name [IN | OUT]
 type [, ...])]  RETURN return_datatype
IS
  [declaration_section]
BEGIN
  < function_body >
END ;
```

Here:

- ✓ **Function_name:** specifies the name of the function.
- ✓ **[OR REPLACE]** option allows replaces an existing function.
- ✓ The **optional parameter list** contains name, mode and types of the parameters.
- ✓ **IS** marks the beginning of the body of the function and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section.
- ✓ **IN** indicates parameter value passed to the function. It is read only value.
- ✓ **OUT it indicates parameter value returned to the user.**

The function must contain a return statement.

- ✓ RETURN clause specifies the header section defines the return type of the function. The return data type can be any of the oracle data type like varchar, number etc.

- **PL/SQL Function Example**

Let's see a simple example to **create a function**.

```
SQL>edit fun1
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/
```

- **Execute Function**
  After write the PL/SQL function you need to execute the function.
  SQL>@fun1
  Function created.
  PL/SQL procedure successfully completed.

  Now write another program to **call the function**.

```
DECLARE
  n3 number(2);
BEGIN
  n3 := adder(11,22);
  dbms_output.put_line('Addition is: ' || n3);
END;
/
```

- **PL/SQL Program Result**
  SQL>@fun
  Output:
  Addition is: 33

- **PL/SQL Drop Function**
  You can drop PL/SQL function using DROP FUNCTION statements.
  - **Functions Drop Syntax**
    DROP FUNCTION function_name;
  - **Functions Drop Example**
    SQL>DROP FUNCTION fun1;
    Function dropped.

❖ TRIGGER

- **What is a Trigger?**
  A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table.
  Triggers are stored programs, which are automatically executed or fired when some events occur.
  A trigger is triggered automatically when an associated DML statement is executed.

- **Types of PL/SQL Triggers**
  There are two types of triggers based on which level it is triggered.
  1) Row level trigger - An event is triggered for each row updated, inserted or deleted.
  2) Statement level trigger - An event is triggered for each SQL statement executed.

- **BEFORE Trigger:** BEFORE trigger execute before the triggering DML statement (INSERT, UPDATE, DELETE) execute. Triggering SQL statement is may or may not execute, depending on the BEFORE trigger conditions block.
- **AFTER Trigger**: AFTER trigger execute after the triggering DML statement (INSERT, UPDATE, DELETE) executed. Triggering SQL statement is executed as soon as followed by the code of trigger before performing Database operation.

- **Component of Trigger**
  1. **Triggering SQL statement**: SQL DML (INSERT, UPDATE and DELETE) statement that execute and implicitly called trigger to execute.
  2. **Trigger Action**: When the triggering SQL statement is execute, trigger automatically call and PL/SQL trigger block execute.
  3. **Trigger Restriction:** We can specify the condition inside trigger to when trigger is fire.

- **Creating a trigger:**
  - **Syntax for creating trigger:**

```
CREATE [OR REPLACE] TRIGGER trigger_name {BEFORE | AFTER}
{INSERT [OR] | UPDATE [OR] | DELETE}  [OF col_name]
ON table_name
 [FOR EACH ROW [ WHEN (condition) ]]
 [REFERENCING OLD AS o NEW AS n]
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Here,

- ✓ **CREATE [OR REPLACE] TRIGGER trigger_name**: It creates or replaces an existing trigger with the trigger_name.
- ✓ **{BEFORE | AFTER}**: This specifies when the trigger would be executed.
- ✓ **{INSERT [OR] | UPDATE [OR] | DELETE}:** This specifies the DML operation.
- ✓ **[OF col_name]:** This specifies the column name that would be updated.
- ✓ **[ON table_name]:** This specifies the name of the table associated with the trigger.
- ✓ **[REFERENCING OLD AS OLD NEW AS NEW]:** This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- ✓ **[FOR EACH ROW]:** This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.

- ✓ **WHEN (condition):** This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

- **Create trigger Example:**
  Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;

BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

  Trigger created.

- Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table:
  - ✓ INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)VALUES (7, 'Rudri', 22, 'HP', 7500.00);
- When a record is created in CUSTOMERS table, above create trigger **display_salary_changes** will be fired and it will display the following result:
  Old salary:
  New salary: 7500
  Salary difference:
- Because this is a new record so old salary is not available and above result is coming as null. Now, let us perform one more DML operation on the CUSTOMERS table. Here is one UPDATE statement, which will update an existing record in the table:
  UPDATE customers SET salary = salary + 500 WHERE id = 2;
- When a record is updated in CUSTOMERS table, above create trigger **display_salary_changes** will be fired and it will display the following result:
  Old salary: 1500
  New salary: 2000
  Salary difference: 500

❖ DATA CONCURRENCY AND LOCKING :

➢ Data Concurrency:

Data concurrency allows many users to access the same data at the same time. One of the primary concerns of a multi-user database system like oracle is how to control concurrency. One way of managing data concurrency to make other users wait until their turn comes. The goal of the database system is to reduce wait time. But at the same time data integrity cannot be compromised. Oracle uses multi versioning locking mechanism to increase data concurrency while maintaining data integrity.

➢ Introduction to Locking

Oracle database allows the user to employ lock on transaction processing by which oracle prevents data from being changed by more than one user at a time. That is it derives interactions which leads to incorrect data or incorrect alteration to data structure.

Oracle locking is fully automatic; there is no need to take extra care by users. When lock is applied, no data can be read while it is being updated while it is being read and no data can be read while it is being updated. Hence only a single user can control the data at any given instance of time.

There are two types of lock:

1. **Shared Locks:**
   Shared locks are placed on resources whenever a Read operation is performed.
   Multiple shared locks can be simultaneously set on a resource
2. **Exclusive Locks:**
   Exclusive locks are placed on resources whenever Write operations are performed.
   Only one exclusive lock can be placed on a resource at a time.

There are two different mode of locking:

1. **Implicit Locking:**
   The oracle DBA applies this lock automatically. The type of lock is applied, corresponding to execution of query. Write operation set exclusive locks, read operation set shared locks. Oracle applies two level of automatic locking table level and page level. Types of lock oracle applies is based upon whether a where condition is used in SQL statement or not.
2. **Explicit Locking:**
   When locking is defined by user it is called explicit locking. Explicit lock can be defined by DBA, owner of table or by user who have been granted privileges. User can explicitly lock either table or row.

❖ PACKAGE :

PL/SQL package is a logical grouping of a related subprogram (procedure/function) into a single element. A Package is compiled and stored as a database object that can be used later.
PL/SQL package has two components.

1) Package Specification
2) Package Body

1) Package Specification

Package specification consists of a declaration of all the public variables, cursors, objects, procedures, functions, and exception. The elements which are all declared in the specification can be accessed from outside of the package. Such elements are known as a public element.

Syntax

```
CREATE [OR REPLACE] PACKAGE <package_name>
{IS|AS}
                    PL/SQL Package specification
END;
```

2) Package Body

It consists of the definition of all the elements that are present in the package specification. It can also have a definition of elements that are not declared in the specification; these elements are called private elements and can be called only from inside the package.

Syntax:

```
CREATE [OR REPLACE] PACKAGE BODY <package_name>
{IS|AS}
        PL/SQL Package body
END;
```

Package body contains:

- ✓ It should contain definitions for all the subprograms/cursors that have been declared in the specification.
- ✓ It can also have more subprograms or other elements that are not declared in specification. These are called private elements.
- ✓ It is a dependable object, and it depends on package specification.
- ✓ The private elements should be defined first before they are used in the package body.
- ✓ The first part of the package is the global declaration part. This includes variables, cursors and private elements (forward declaration) that is visible to the entire package.
- ✓ The last part of the package is Package initialization part that executes one time whenever a package is referred first time in the session.
- ✓ **PL/SQL Package Example**
- ✓ PL/SQL Package example step by step explain to you, you are creating your own package using this reference example. We have emp1 table having employee information,

EMP_NO	EMP_NAME	EMP_DEPT	EMP_SALARY
1	Forbs ross	Web Developer	45k
2	marks jems	Program Developer	38k
3	Saulin	Program Developer	34k
4	Zenia Sroll	Web Developer	42k

- **Package Specification**

Create Package specification code for defining procedure, function IN or OUT parameter and execute package specification program.

```
CREATE or REPLACE PACKAGE pkg1
 IS | AS
   PROCEDURE pro1 (no in number, name out varchar2);
   FUNCTION fun1 (no in number) RETURN varchar2;
   END;
/
```

- **Package Body**

Create Package body code for implementing procedure or function that is defined package specification. Once you implement execute this program.

```
CREATE or REPLACE PACKAGE BODY pkg1
IS
   PROCEDURE pro1(no in number,info our varchar2)
      IS
      BEGIN
         SELECT * INTO temp FROM emp1 WHERE eno = no;
      END;

   FUNCTION fun1(no in number) return varchar2
      IS
      name varchar2(20);
      BEGIN
         SELECT ename INTO name FROM emp1 WHERE eno = no;
         RETURN name;
      END;
END;
/
```

- **Pl/SQL Program calling Package**

Now we have a one package pkg1, to call package defined function, procedures also pass the parameter and get the return result.

pkg_prg.sql

```
DECLARE
   no number := &no;
   name varchar2(20);
BEGIN
   pkg1.pro1(no,info);
   dbms_output.put_line('Procedure Result');
```

```
    dbms_output.put_line(info.eno||'     '||
                         info.ename||'   '||
                         info.edept||'   '||
                         info.esalary||' '||);
    dbms_output.put_line('Function Result');
    name := pkg1.fun1(no);
    dbms_output.put_line(name);
END;
/
```

- **Result**

  Now execute the above created pkg_prg.sql program to asking which user information you want to get, you put user id and give information.

  SQL>@pkg_prg

```
no number &n=2
Procedure Result
2    marks jems    Program Developer    38K
Function Result
marks jems

PL/SQL procedure successfully completed.
```

- **PL/SQL Package Drop**

  You can drop package using package DROP statement,

  Syntax

  DROP PACKAGE package_name;

  Example

  SQL>DROP PACKAGE pkg1;

  Package dropped.