PL/SQL

- PL/SQL (Procedural Language/Structured Query Language) program executes on Oracle database.
- PL/SQL includes procedural language elements (like declaration, conditions, looping), also, you can able to handle a runtime exception.

- What is PL/SQL?
- PL/SQL is a Procedural Language extension of Structured Query Language (SQL). PL/SQL is specially designed for Database oriented activities. Oracle PL/SQL allows you to perform data manipulation operation those are safe and flexible.
- PL/SQL is a very secure functionality tool for manipulating, controlling, validating, and restricting unauthorized access data from the SQL database.
- Using PL/SQL we can improve application performance. It also allows to deal with errors so we can provide user friendly error messages.
- PL/SQL have a great functionality to display multiple records from the multiple tables at the same time.
- PL/SQL is capable to send entire block of statements and execute it in the Oracle engine at once.

 Advantages PL/SQL

- Procedural language support : PL/SQL is a development tools not only for data manipulation futures but also provide the conditional checking, looping or branching operations same as like other programming language.
- Reduces network traffic : This one is great advantages of PL/SQL. Because PL/SQL nature is entire block of SQL statements execute into oracle engine all at once so it's main benefit is reducing the network traffic.
- Error handling : PL/SQL is dealing with error handling, It's permits the smart way handling the errors and giving user friendly error messages, when the errors are encountered.
- Declare variable : PL/SQL gives you control to declare variables and access them within the block. The declared variables can be used at the time of query processing.
- Intermediate Calculation : Calculations in PL/SQL done quickly and efficiently without using Oracle engines. This improves the transaction performance.
- Portable application : Applications are written in PL/SQL are portable in any Operating system. PL/SQL applications are independence program to run any computer.

 PL/SQL Block Structure

- PL/SQL is block structured language divided into three logical blocks.
- BEGIN block and END; keyword are compulsory, and block EXCEPTION are optional block. END; is not a block only keyword to end of PL/SQL program.
- PL/SQL block structure follows divide-and-conquer approach to solve the problem stepwise.

- DECLARE

- Variables and constants are declared, initialized within this section.
- Variables and Constants : In this block, declare and initialize variables (and constants). You must have to declare variables and constants in declarative block before referencing them in procedural statement.
- Declare Variables and Assigning values : You can define variable name, data type of a variable and its size. Date type can be: CHAR, VARCHAR2, DATE, NUMBER, INT or any other.
- Declare Constants and Assigning values : Constants are declared same as variable but you have to add the CONSTANT keyword before defining data type. Once you define a constant value you can't change the value.

- BEGIN

- BEGIN block is procedural statement block which will implement the actual programming logic. This section contains conditional statements (if...else), looping statements (for, while) and Branching Statements (goto) etc.

- EXCEPTION

- PL/SQL easily detects user defined or predefined error condition. PL/SQL is famous for handling errors in smart way by giving suitable user friendly messages. Errors can be rise due to wrong syntax, bad logical or not passing a validation rules.
- You can also define exception in your declarative block and later you can execute it by RAISE statement.
- Note :
- BEGIN block and END; keyword are compulsory of any PL/SQL program.
- Where as EXCEPTION block are optional.

```
DECLARE -- DECLARE block, declare and initialize values
designation VARCHAR2(30);
enonumber(5) := 5;        id BOOLEAN;
inter INTERVAL YEAR(2) TO MONTH;
BEGIN -- BEGIN block, also assign values
designation := UPPER('Web Developer');
id := TRUE;       inter := INTERVAL '45'
YEAR;
END;
/
```

- PL/SQL Data Types

1. Scalar data types - Scalar data types haven't internal components.
2. LOB data types - Stores large objects such as images, graphics, video.
3. Unknown Column types - Identify columns when not know type of column.

- Scalar types
- Scalar data type haven't internal components. It is like a linear data type. Scales data type divides into four different types character, numeric, boolean or date/time type.

Data types	Description, Storage(Maximum)	
NUMBER(p,s)	NUMBER data type used to store numeric data. It's contain letters, numbers, and special characters. Storage Range : Precision range(p) : 1 to 38 and Scale range(s) : -84 to 127 NUMBER Subtypes : This sub type defines different types storage range.	
Sub Data types	**Maximum Precision**	**Description**
INTEGER	38 digits	This data types are used to store fixed decimal points. You can use based on your requirements.
INT	38 digits	
SMALLINT	38 digits	
DEC	38 digits	
DECIMAL	38 digits	
NUMERIC	38 digits	
REAL	63 binary digits	
DOUBLE PRECISION	126 binary digits	
FLOAT	126 binary digits	

Data types	Description	Storage(Maximum)
CHAR	CHAR data type used to store character data within predefined length.	32767 bytes
CHARACTER	CHARACTER data type same as CHAR type, is this another name of CHAR type.	32767 bytes
VARCHAR2	VARCHAR2 data type used to store variable strings data within predefined length. VARCHAR2 Subtypes : Following sub type defines same length value.	
Sub Data types	Description	
STRING	We can access this data type.	
VARCHAR		32767 bytes
NCHAR	NCHAR data type used to store national character data within predefined length.	32767 bytes
NVARCHAR2	NVARCHAR2 data type used to store Unicode string data within predefined length.	32767 bytes
RAW	The RAW data type used to store binary data such as images, graphics etc.	32767 bytes
LONG	LONG data type used to store variable string data within predefined length, This data type used for backward compatibility. Please use LONG data to the CLOB type.	32760 bytes
LONG RAW	LONG RAW data type same as LONG type used to store variable string data within predefined length, This data type used for backward compatibility. Use LONG RAW data type for storing BLOB type data.	32760 bytes
ROWID	The ROWID data type represents the actual storage address of a row. And table index identities as a logical rowid. This data type used to storing backward compatibility.	

- Boolean Data Type
- Boolean Data types stores logical values either TRUE or FALSE. Let's see Boolean data types in PL/SQL:

Data types	Description
Boolean	Boolean data type stores logical values. Boolean data types doesn't take any parameters. Boolean data type store either TRUE or FALSE. Also store NULL, Oracle treats NULL as an unassigned boolean variable. You can not fetch boolean column value from another table.

•Date/Time Data types

•Date/time variable can holds value, we can say date/time data type. PL/SQL automatically converts character value in to default date format ('DD-MM-YY') value.

•Following are the Date/Time data types in PL/SQL:

Data types	Description	Range
DATE	DATE data type stores valid date-time format with fixed length. Starting date from Jan 1, 4712 BC to Dec 31, 9999 AD.	Jan 1, 4712 BC to Dec 31, 9999 AD

LOB types

LOB data types used to store large objects such as image, video, graphics, text or audio. Maximum size is up to 4 Gigabytes. Following are the LOB data types in PL/SQL:

Data types	Description	Storage(Maximum)
BFILE	BFILE data type is used to store large binary objects into Operating System file. BFILE stores full file locator's path which are points to a stored binary object with in server. BFILE data type is read only, you can't modify them.	Size: up to 4GB (2^{32} - 1 bytes) Directory name: 30 character File name: 255 characters
BLOB	BLOB data type is same as BFILE data type used to store unstructured binary object into Operating System file. BLOB type fully supported transactions are recoverable and replicated.	Size: 8 TB to 128 TB (4GB - 1) * DB_BLOCK_SIZE
CLOB	CLOB data type is used to store large blocks of character data into Database. Store single byte and multi byte character data. CLOB type is fully supported transactions, also that are recoverable and replicated.	Size: 8 TB to 128 TB (4GB - 1) * DB_BLOCK_SIZE

Unknown Column types

- PL/SQL this data type is used when column type is not know.

Data types	Description
%Type	This data type is used to store value unknown data type column in a table. Column is identified by %type data type. Eg. emp.eno%type emp name is table, eno is a unknown data type column and %Type is data type to hold the value.
%RowType	This data type is used to store values unknown data type in all columns in a table. All columns are identified by %RowType datatype. Eg. emp%rowtype emp name is table, all column type is %rowtype.
%RowID	RowID is data type. RowID is two type extended or restricted. Extended return 0 and restricted return 1 otherwise return the row number. Function of Row ID:

Variable Declaration Syntax

variable_nameDatatype[Size] [NOT NULL] := [value];

- Explanation:
- variable_name is the predefined name of the variable.
- Data type is a valid PL/SQL data type.
- Size is an optional specification of data type size to hold the maximum size value.
- NOT NULL is an optional specification of the variable value can't be accept NULL.
- value is also an optional specification, where you can initialize the initial value of variable.
- Each variable declaration is terminated by a semicolon.

 Example

```
DECLARE
enonumber(5) NOT NULL := 2; -- NOT NULL (value can't be blank), Assign initial value
ename varchar2(15) := 'Branson Devs'; -- intialize value at the time of declaration
BEGIN
dbms_output.put_line('Declared Value:'); dbms_output.put_line('
Employee Number: ' || eno || ' Employee Name: ' || ename);
END;
/
```

 PL/SQL Constant Declaration

Syntax

Constant_name CONSTANT Datatype[Size] := Value; Explanation:

- Constant_name is a predefined name of the constant (similar to a variable name).
- CONSTANT is a reserved keyword.
- Data type is a valid PL/SQL data type.
- Size is an optional specification of data type. It holds maximum capacity value for the particular variable.
- Value must be assigned to a constant when it is declared. You can not assign or change it later.
- Each constant declaration is terminated by a semicolon.

- ExampleDECLARE

```
pi CONSTANT REAL := 3.14159;
radius REAL := 3;
area REAL := (pi * radius**2);
BEGIN
dbms_output.put_line(' PI: ' || pi || ' Radius: ' || radius);          dbms_output.put_line('
Area: ' || area);
END;
/
```

PL/SQL SET Serveroutput ON

Whenever you start Oracle SQL (PL/SQL) at that time you must have to write the "SET Serveroutput ON" command.

PL/SQL program execution into Oracle engine so we always required to get serveroutput result and display into the screen otherwise result can't be display.

Example

```
set serveroutput on
DECLARE
enonumber(5) NOT NULL := 2        ename
varchar2(15) := 'Branson Devs';
edept CONSTANT varchar2(15) := 'Web Developer';  BEGIN
dbms_output.put_line('Declared Value:');        dbms_output.put_line(' Employee
Number: ' || eno || ' Employee   Name: ' || ename);        dbms_output.put_line('Constant
Declared:');
dbms_output.put_line(' Employee Department: ' || edept);  END;
/
```

PL/SQL Comments

- PL/SQL Comments you can write single line comments or either multiple line comments,
- Multi-line comments and Single line comments
- Multi-line comments are delimited with /*..COMMENT TEXT..*/
- single line comments starts with two dashes --.

- PL/SQL Comments can begin in any column on any line. If you are embedding comments in SQL that will be embedded in PL/SQL you need to be careful for writing a column.

Control statement

- PL/SQL IF THEN ELSE conditional control statements. PL/SQL Conditional Control two type: IF THEN ELSE statement and CASE statement,
- PL/SQL IF statement check condition and transfer the execution flow on that matched block depending on a condition. IF statement execute or skip a sequence of one or more statements. PL/SQL IF statement four different type,

1. IF THEN Statement
2. IF THEN ELSE Statement
3. IF THEN ELSIF Statement
4. Nested IF THEN ELSE Statement

1.If..then

Syntax:

```
IF ( condition )
THEN  statement
END IF;
```

example:

```
 DECLARE
no INTEGER(2) := 14;
BEGIN
    IF ( no = 14 ) THEN
       DBMS_OUTPUT.PUT_LINE('condition true');
    END IF;
END;
/
```

2.IF THEN ELSE Statement

Syntax:

```
     IF ( condition ) THEN
statement;       ELSE
statement;
    END IF;
```

Example:

```
DECLARE
      no INTEGER(2) := 14;
BEGIN
      IF ( no = 11 ) THEN
            DBMS_OUTPUT.PUT_LINE(no || ' is same');
     ELSE
          DBMS_OUTPUT.PUT_LINE(no || ' is not same');
      END IF;
END;
/
```

3.IF THEN ELSIF Statement

Syntax:

```
IF ( condition-1 ) THEN
statement-1;
ELSIF ( condition-2 ) THEN
        statement-2;
ELSIF ( condition-3 ) THEN
   statement-3;
ELSE
         statement;
END IF;
```

Example:

```
DECLARE
        result CHAR(20) := 'second';
BEGIN
       IF ( result = 'distinction' ) THEN
              DBMS_OUTPUT.PUT_LINE('First Class with  Distinction');
       ELSIF ( result = 'first' ) THEN
               DBMS_OUTPUT.PUT_LINE('First Class');
       ELSIF ( result = 'second' ) THEN
              DBMS_OUTPUT.PUT_LINE('Second Class');
       ELSIF ( result = 'third' ) THEN
              DBMS_OUTPUT.PUT_LINE('Third Class');
       ELSE
              DBMS_OUTPUT.PUT_LINE('Fail');
      END IF;
END;
/
```

4.Nested IF THEN ELSE Statement syntax

```
IF ( condition-1 ) THEN
statement-1;
  ELSE IF ( condition-2 ) THEN
statement-2;
      ELSE IF ( condition-3 ) THEN
statements-3;
      END IF;
   END IF;
END IF;
```

Example

```
DECLARE
     gender CHAR(20) := 'female';
      result CHAR(20) := 'second';
BEGIN
          IF ( gender = 'male' ) THEN
            DBMS_OUTPUT.PUT_LINE('Gender Male Record Skip!');
          ELSE IF ( result = 'distinction' ) THEN
             DBMS_OUTPUT.PUT_LINE('First Class with Distinction');
          ELSIF ( result = 'first' ) THEN
              DBMS_OUTPUT.PUT_LINE('First Class');
          ELSIF ( result = 'second' ) THEN
               DBMS_OUTPUT.PUT_LINE('Second Class');
          ELSIF ( result = 'third' ) THEN
                 DBMS_OUTPUT.PUT_LINE('Third Class');
          ELSE
```

```
            DBMS_OUTPUT.PUT_LINE('Fail');
          END IF;
        END IF;
END;
```

- PL/SQL Loop - Basic Loop, FOR Loop, WHILE Loop syntax

```
  [ label_name ] LOOP
statement(s);
   END LOOP [ label_name ];
```

Example

```
DECLARE
   no NUMBER := 5;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Inside value: no = ' || no);
no := no -1;          IF no = 0 THEN
            EXIT;
    END IF;
      DBMS_OUTPUT.PUT_LINE ('Inside value: no = ' || no);
  END LOOP;
    DBMS_OUTPUT.PUT_LINE('Outside loop end');
END;
/
```

While Loop syntax

```
  [ label_name ] WHILE condition LOOP
statement(s);
   END LOOP [ label_name ];
```

ExampleDECLARE

```
    no NUMBER := 0;
BEGIN
  WHILE no < 10
LOOP
no := no + 1;
      DBMS_OUTPUT.PUT_LINE('Sum :' || no);
   END LOOP;
      DBMS_OUTPUT.PUT_LINE('Sum :' || no);
END;
/
```

For Loop

```
[ label_name ] FOR current_value IN [ REVERSE ] lower_value..upper_value
LOOP
   statement(s);
END LOOP [ label_name ];
```

Example

```
BEGIN
FOR no IN 1 .. 5
LOOP
      DBMS_OUTPUT.PUT_LINE('Iteration : ' || no);
END LOOP;
END;
/
BEGIN
FOR no IN  reverse 1 .. 5
LOOP
      DBMS_OUTPUT.PUT_LINE('Iteration : ' || no);
END LOOP;
END;
/
```

PL/SQL EXIT CONTINUE GOTO Statements

Exit Statement syntax

```
[ label_name ] LOOP
     statement(s);
     EXIT;
END LOOP [ label_name ];
```

Example

```
DECLARE
      no NUMBER := 5;
BEGIN
  LOOP
       DBMS_OUTPUT.PUT_LINE ('Inside value: no = ' || no);
no := no -1;          IF no = 0 THEN
            EXIT;
       END IF;
  END LOOP;
       DBMS_OUTPUT.PUT_LINE('Outside loop end'); -- After EXIT control transfer this
statement
END;
```

Exit when statementsystax

```
[ label_name ] LOOP
       statement(s);
      EXIT WHEN condition;
END LOOP [ label_name ];
```

ExampleDECLARE

```
       i number;
BEGIN
LOOP  dbms_output.put_line('Hello');
          i:=i+1;
          EXIT WHEN i>5;
      END LOOP;
END;
/
```

Continue statement

syntax IF condition

```
THEN
       CONTINUE;
END IF;
```

Example

```
DECLARE
      no NUMBER := 0;
BEGIN
      FOR no IN 1 .. 5 LOOP
            IF i = 4 THEN
                CONTINUE;
            END IF;
              DBMS_OUTPUT.PUT_LINE('Iteration : ' || no);
       END LOOP;
END;
/
```

GOTO Statement

syntax GOTO

```
code_name
-----------
-----------
<<code_name>>
-----------
-----------
```

Example

```
BEGIN
FOR i IN 1..5 LOOP
                    dbms_output.put_line(i);
```

```
IF i=4 THEN  GOTO
          label1;
END IF;
          END LOOP;
          <<label1>> DBMS_OUTPUT.PUT_LINE('Row Filled');  END;
```

Case Statement syntax

```
    CASE selector
         WHEN value-1 THEN
            statement-1;
         WHEN value-2 THEN
            statement-2;
ELSE
statement-3;
      END CASE
```

Example

```
DECLARE          a
number := 7;
BEGIN
          CASE a
    WHEN 1 THEN
          DBMS_OUTPUT.PUT_LINE('value 1');
   WHEN 2 THEN
          DBMS_OUTPUT.PUT_LINE('value 2');
   WHEN 3 THEN
          DBMS_OUTPUT.PUT_LINE('value 3');
   ELSE
          DBMS_OUTPUT.PUT_LINE('no matching CASE found');
     END CASE;
END;
/
```

CURSOR

- What is Cursor?
- Cursor is the work area which Oracle reserves for internal processing of SQL statements. This work area is private for oracles reserved are called cursor.
- How to Use Cursor
- In PL/SQL block SELECT statement can not return more than one row at a time. So Cursor use to some group of rows (more than one row) for implementing certain logic to get all the group of records.
- What is Active-data-set ?
- Cursor to store data in work area this work area is called active data set.
- Cursors can be classified as:
- Implicit Cursor or Internal Cursor : Manage for Oracle itself or internal process itself.
- Explicit Cursor or User-defined Cursor : Manage for user/programmer or external processing.

implicit cursor v/s explicit cursor

- The main difference between the implicit cursor and explicit cursor is that an explicit cursor needs to be defined explicitly by providing a name while implicit cursors are automatically created when you issue a select statement.
- multiple rows can be fetched using explicit cursors while implicit cursors can only fetch a single row.
- Also NO_DATA_FOUND and TOO_MANY_ROWS exceptions are not raised when using explicit cursors , as opposed to implicit cursors.
- implicit cursors are more vulnerable to data errors and provide less programmatic control than explicit cursors.
- implicit cursors are considered less efficient than explicit cursors.

Implicit Cursor

- Oracle uses implicit cursors for its internal processing. Even if we execute a SELECT statement or DML statement Oracle reserves a private SQL area in memory called cursor.
- Implicit cursor scope you can get information from cursor by using session attributes until another SELECT statement or DML statement execute.

Cursor Attributes

Cursor Attribute	Cursor Variable	Description
%ISOPEN	SQL%ISOPEN	Oracle engine automatically open the cursor If cursor open return TRUE otherwise return FALSE.
%FOUND	SQL%FOUND	If SELECT statement return one or more rows or DML statement (INSERT, UPDATE, DELETE) affect one or more rows If affect return TRUE otherwise return FALSE. If not execute SELECT or DML statement return NULL.
%NOTFOUND	SQL%NOTFOUND	If SELECT INTO statement return no rows and fire no_data_found PL/SQL exception before you can check SQL%NOTFOUND. If not affect the row return TRUE otherwise return FALSE.
%ROWCOUNT	SQL%ROWCOUNT	Return the number of rows affected by a SELECT statement or DML statement (insert, update, delete). If not execute SELECT or DML statement return NULL.

Implicit cursor example

```
SQL>set serveroutput on
SQL>BEGIN
   UPDATE emp_information SET emp_dept='Web Developer' WHERE emp_name='Saulin';
      IF SQL%FOUND THEN
dbms_output.put_line('Updated - If Found');
      END IF;
```

```
    IF SQL%NOTFOUND THEN
dbms_output.put_line('NOT Updated - If NOT Found');
    END IF;
    IF SQL%ROWCOUNT>0 THEN
dbms_output.put_line(SQL%ROWCOUNT||' Rows Updated');      ELSE
dbms_output.put_line('NO Rows Updated Found');
  END;
 /
```

Explicit cursor

- Explicit Cursor which are construct/manage by user itself call explicit cursor.
- User itself to declare the cursor, open cursor to reserve the memory and populate data, fetch the records from the active data set one at a time, apply logic and last close the cursor.
- You can not directly assign value to an explicit cursor variable you have to use expression or create subprogram for assign value to explicit cursor variable.

Step for using Explicit Cursor

1.Declare cursor

- CURSOR cursor_name [parameter] [RETURN return_type] IS SELECT STATEMENT;

2.Opening Explicit Cursor

- OPEN cursor_name[(cursor_parameter)];

3.Loop

4.Fetching data from cursor

- FETCH cursor_name INTO variable;

5.Exit loop

6.Closing Explicit Cursor

- CLOSE cursor_name[(cursor_parameter)];

Example

```
SQL>set serveroutput on
SQL>DECLARE
     cursor c is select * from emp;
tmpemp%rowtype;
    BEGIN
       OPEN c;
Loop  exit when c%NOTFOUND;
            FETCH c into tmp;
                update emp set salary = 50000 where eno = 1;
          END Loop;
             IF c%ROWCOUNT>0 THEN
dbms_output.put_line(SQL%ROWCOUNT||' Rows Updated');
            ELSE
dbms_output.put_line('NO Rows Updated Found');
             END IF;
```

```
    CLOSE c;
END;
/
```

Example

The Bank Manager has Decided to mark all those accounts as inactive(I) on which there are no transactions performed in the last 365 days. Whenever any such update takes place, a record for the same is maintained in the INACTV_ACCT_MSTR table comprising of the account number, the opening date and type of account write a PL/SQL block using Explicit cursor.

Acct_mstr(acct_no,status,opndt,type)
Trans_mst(trans_no,acct_no)
INACTV_ACCT_MSTR(acct_no,opndt,type)

```
create table acc_mst
(acc_no integer primary key,status char(5),opendtdate,type char(10));

create table tran_mst
(tran_no integer primary key,acc_no integer references acc_mst(acct_no));
create table inact_acc_mst
(sacc_nointeger,sopendtdate,stype char(10));

insert into acc_mstvalues(1,'w','20-feb-2010','current');
insert into acc_mstvalues(2,'r','3-jan-2009','saving'); insert
into acc_mstvalues(3,'s','12-may-2010','current'); insert
into acc_mstvalues(4,'a','11-jun-2011','fixdeposit'); insert
into acc_mst values(5,'s','29-nov-2012','saving');

insert into tran_mstvalues(10,1); insert
into tran_mst values(20,2); insert into
tran_mst values(30,3); insert into
tran_mst values(40,4); insert into
tran_mst values(50,5);

set serveroutput on declare
cursor c1is   select
acc_no,status,opendt,type     from
acc_mst
          where acc_noin(select  acc_no
                         from tran_mst
                         group by acc_no
               having max(sysdate - opendt)>365);
      sacc_no number;          sstatus varchar(20);      sopendt
date;
          stypevarchar(20);
```

```
begin
        open c1;
if c1 %isopenthen    loop
        fetch c1 into sacc_no,sstatus,sopendt,stype;
exit when c1 %notfound;
      if c1 %found then
            update acc_mst
      set status='s'
                        where acc_no=sacc_no;
            insert into inact_acc_mst

      values(sacc_no,sopendt,stype);
      end if;  end loop;
        else
dbms_output.put_line('unable to open cursor');       end if;
close c1; end; /
```

Cursor For Loop

```
SQL>set serveroutput on
SQL>DECLARE
        cursor c1 is select * from emp ;
tmpemp%rowtype;
BEGIN

        FOR tmp IN c1
       LOOP

    update emp set salary= 20000;
      END Loop;
END;
/
```

Example

The Bank Manager has Decided to mark all those accounts as inactive(I) on which there are no transactions performed in the last 365 days. Whenever any such update takes place, a record for the same is maintained in the INACTV_ACCT_MSTR table comprising of the account number, the opening date and type of account write a PL/SQL block using Explicit cursor.

Acct_mstr(acct_no,status,opndt,type)
Trans_mst(trans_no,acct_no)
INACTV_ACCT_MSTR(acct_no,opndt,type)

```
create table acc_mst(acc_no integer primary key,status char(5),opendtdate,type char(10)); create
table tran_mst(tran_no integer primary key,acc_no integer references acc_mst(acct_no)); create
table inact_acc_mst(sacc_nointeger,sopendtdate,stype char(10));
```

```
insert into acc_mstvalues(1,'w','20-feb-2010','current');
insert into acc_mstvalues(2,'r','3-jan-2009','saving'); insert
into acc_mstvalues(3,'s','12-may-2010','current'); insert
into acc_mstvalues(4,'a','11-jun-2011','fixdeposit'); insert
into acc_mst values(5,'s','29-nov-2012','saving');

insert into tran_mstvalues(10,1); insert
into tran_mst values(20,2); insert into
tran_mst values(30,3); insert into
tran_mst values(40,4); insert into
tran_mst values(50,5);

set serveroutput on declare
          cursor c1 is
           select acct_no,status,opendt,type
          from acc
           where acct_noin(select  acct_no from trans
                          group by acct_no
               having max(sysdate-opendt)>365); begin
          for c in c1
loop
       update acc
  set status='s'    where acct_no=c.acct_no;    insert
into inact_acc   values(c.acct_no,c.opendt,c.type);
          end loop;
end;
/
```

Parameterized Cursor

- PL/SQL Parameterized cursor pass the parameters into a cursor and use them in to query.
- PL/SQL Parameterized cursor define only datatype of parameter and not need to define it's length.
- Default values is assigned to the Cursor parameters. and scope of the parameters are locally.
- Parameterized cursors are also saying static cursors that can passed parameter value when cursor are opened.

Example

```
SQL>set serveroutput on
SQL>DECLARE
cursor  test(no number) is select * from emp where eno = no;
tmpemp%rowtype;
```

```
        BEGIN
            FOR tmpIN  test(4)
            LOOP
dbms_output.put_line('EMP_No: '||tmp.eno);
dbms_output.put_line('EMP_Name:'||tmp.ename);
dbms_output.put_line('EMP_Salary:'||tmp.salary);
             END Loop;
END;
/
```

Cursor Within Cursor [Nested Cursor]

```
declare    cursor c_emp is select *
from emp;    cursor c_dept is select * from
dept; begin     for c1_emp in c_emp
loop        for c1_dept in c_dept         loop
          insert  into  emp_dept  values(c1_emp.eno,c1_emp.ename,c1_emp.salary,c1_dept.dname);
end loop;     end loop;
end;
```

Nested cursor (to generate report departmentwise wise employeelist and also print total salary)

```
declare
 cursor dept_curis  selectd_id,d_name  from tbldept;  cursor emp_cur is
select e_id,e_name,e_salary,d_id from tblemp;
dept_recdept_cur%rowtype;
emp_recemp_cur%rowtype;
tot_salnumber(10):=0; begin
       for dept_rec in dept_cur
          loop
           dbms_output.put_line(rpad(dept_rec.d_id,15)||' '||rpad(dept_rec.d_name,15));

             for emp_rec in emp_cur
loop                        if dept_rec.d_id=emp_rec.d_id
then
tot_sal:=tot_sal+emp_rec.e_salary;  dbms_output.put_line(rpad(emp_rec.e_id,15)||'  '||rpad
(emp_rec.e_name,15)||'
'||rpad(emp_rec.e_salary,15));
       end if;                                                                  end loop;
              dbms_output.put_line('--------------------------------------------------');
          end loop;
       dbms_output.put_line('total salary =>'||tot_sal);
         end;
/
```

Exception Handling

Introduction

- Every PL/SQL block of code encountered by the oracle engine is accepted as a client.
- When SQL statement fails the oracle engine is the first to recognize this as an exception condition.
- The oracle engine immediately tries to handle the exception condition and resolve it.
- This is done by raising a built-in exception handler.
- PL/SQL exceptions are predefined and raised automatically into oracle engine when any error occur during a program.
- Each and every error has defined a unique number and message. When warning/error occur in program it's called an exception to contains information about the error.
- In PL/SQL built in exceptions or you make user define exception. Examples of built-in type (internally) defined exceptions division by zero, out of memory.
- Some common built-in exceptions have predefined names such as ZERO_DIVIDE and STORAGE_ERROR.
- Normally when exception is fire, execution stops and control transfers to the exceptionhandling part of your PL/SQL block. Internal exceptions are raised implicitly (automatically) by the run-time system. User-defined exceptions must be raised explicitly by RAISE statements, which are also raise predefined exceptions.

PL/SQL exceptions consist following three,

- Exception Type
- Error Code
- Error Message

Three Types of Exception

1.Built in exception 2.User named exception 3.User defined exception

1.Buit in Exception

Exception	Error Code	Description
ACCESS_INTO_NULL	ORA-06530	Exception raised when assign uninitialized (NULL) object.
CASE_NOT_FOUND	ORA-06592	Exception raised when no any choice case found in CASE statement as well as no ELSE clause in CASE statement.
CURSOR_ALREADY_OPEN	ORA-06511	Exception raised when you open a cursor that is already opened.
DUP_VAL_ON_INDEX	ORA-00001	Exception raised when you store duplicate value in unique constraint column.
INVALID_CURSOR	ORA-01001	Exception raised when you perform operation on cursor and cursor is not really opened.
INVALID_NUMBER	ORA-01722	Exception raised when you try to explicitly conversion from string to a number fail.
LOGIN_DENIED	ORA-01017	Exception raised when log in into oracle with wrong username or password.
NO_DATA_FOUND	ORA-01403	Exception raised when SELECT ... INTO statement doesn't fetch any row from a database table.
NOT_LOGGED_ON	ORA-01012	Exception raised when your program try to get data from database and actually user not connected to Oracle.
PROGRAM_ERROR	ORA-06501	Exception raised when your program is error prone (internal error).
STORAGE_ERROR	ORA-06500	Exception raised when PL/SQL program runs out of memory or may be memory is dumped/corrupted.
SYS_INVALID_ROWID	ORA-01410	Exception raised when you try to explicitly conversion from string character string to a universal rowid (uid) fail.
TIMEOUT_ON_RESOURCE	ORA-00051	Exception raised when database is locked or ORACLE is waiting for a resource.
TOO_MANY_ROWS	ORA-01422	Exception raised when SELECT ... INTO statement returns more than one row.
VALUE_ERROR	ORA-06502	Exception raised when arithmetic, conversion, defined size constraint error occurs.
ZERO_DIVIDE	ORA-01476	Exception raised when you program try to attempt divide by zero number.

DECLARE

<declarations section>

```
        BEGIN
          <executable command(s)>
        EXCEPTION
<exception handling goes here >        WHEN
exception1 THEN
                  exception1-handling-statements    WHEN
        exception2  THEN        exception2-handling-
        statements.
          WHEN others THEN
                    exception3-handling-statements
END;
/
```

Example

```
DECLARE
emp_recemp%rowtype;
BEGIN
SELECT * INTO emp_rec FROM emp WHERE  eno=1;
dbms_output.put_line(emp_rec.ename);                dbms_output.put_line(emp_rec.salary);
EXCEPTION
        WHEN no_data_found THEN
dbms_output.put_line('Table have not data');            WHEN others
THEN
dbms_output.put_line('Error!');
END;
/
```

Example

```
DECLARE
emp_idtblemp.e_id%type;
      emp_name varchar(10);
      emp_salary number(10);
BEGIN       emp_id := &emp_id;
      emp_name := &emp_name;
         emp_salary := &emp_salary;
       insert into tblemp(e_id,e_name,e_salary) values(emp_id,emp_name,emp_salary);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      DBMS_OUTPUT.PUT_LINE('Duplicate value on an index'); END;
```

User Named Exception

- PL/SQL user named exception. you can define your own error message and error number using Pragma EXCEPTION_INIT or RAISE_APPLICATION_ERROR function.
- PL/SQL pragma EXCEPTION_INIT
- pragma EXCEPTION_INIT : Pragma is a keyword directive to execute proceed at compile time. pragma EXCEPTION_INIT function take this two argument,
- exception_name&error_number

Syntax

- You can define pragrma EXCEPTION_INIT in DECLARE BLOCK on your program.
- PRAGMA EXCEPTION_INIT(exception_name, -error_number);

```
DECLARE
user_define_exception_name EXCEPTION;
    PRAGMA EXCEPTION_INIT(user_define_exception_name,-error_number);
BEGIN
     statement(s);
     EXCEPTION
      WHEN user_define_exception_name THEN
           User defined statement (action) will be taken;
END;
```

Example

```
create table tblproduct(pro_id number(5) primary key,name varchar(10));
insert into tblproduct values(1,'laptop'); insert into tblproduct
values(2,'pendrive'); insert into tblproduct values(3,'ac'); insert into
tblproduct values(4,'tv'); insert into tblproduct values(5,'mobile');
create table tblorder(order_id number(5) primary key,pro_id
    number(5) references tblproduct(pro_id)); insert into tblorder values(101,1); insert into
tblorder values(102,2); insert into tblorder values(103,3); insert into tblorder values(104,4); insert
into tblorder values(105,5);

DECLARE
   Child_rec_exception1 EXCEPTION;
   PRAGMA EXCEPTION_INIT (Child_rec_exception1,-2292);
```

```
BEGIN
      Delete FROM tblproduct where pro_id = 5;
EXCEPTION
    WHEN Child_rec_exception1 THEN
Dbms_output.put_line('Child records are present for this product_id.');  END;
```

User Define Exception

Step For User Define Exception

1) Declare exception

You must have to declare user define exception name in DECLARE block.

user_define_exception_name EXCEPTION;

Exception and Variable both are same way declaring but exception use for store error condition not a storage item.

2) RAISE exception

RAISE statement to raised defined exception name and control transfer to a EXCEPTION block.

```
RAISE user_define_exception_name;
```

3) Implement exception condition

In PL/SQL EXCEPTION block add WHEN condition to implement user define action.

```
WHEN user_define_exception_name THEN
    User defined statement (action) will be taken;
```

Syntax

```
DECLARE
user_define_exception_nameEXCEPTION;  BEGIN
   statement(s);
   IF condition THEN
    RAISE user_define_exception_name;
   END IF;
EXCEPTION
    WHEN user_define_exception_name THEN
     User defined statement (action) will be taken;
END;
```

Example

```
DECLARE
myexEXCEPTION;      i
NUMBER;
BEGIN
  FOR i IN (SELECT * FROM tblemp)
```

```
  LOOP
      IF i.e_id = 103 THEN
          RAISE myex;
      END IF;
 END LOOP;
EXCEPTION
  WHEN myex THEN
dbms_output.put_line('Employee number already exist in tblemp table.'); END;
```

User Named Exception Handlers [business rule validation] Example

- create table customers(id number(5),name varchar(10),address varchar(10));
- insert into customers values(1,'aaa','surat');
- insert into customers values(2,'bbb','baroda');
- insert into customers values(3,'ccc','bharuch');

```
DECLARE
c_idcustomers.id%type       :=       &cc_id;
c_namecustomers.name%type;
c_addrcustomers.address%type;
ex_invalid_id  EXCEPTION;
BEGIN
          IF c_id<= 0 THEN
             RAISE ex_invalid_id;
          ELSE
SELECT  name, address INTO  c_name, c_addr FROM customers  WHERE id = c_id;

             DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
             DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
         END IF;
EXCEPTION
WHEN ex_invalid_id THEN
dbms_output.put_line('ID must be greater than zero!');              WHEN
no_data_found THEN
            dbms_output.put_line('No such customer!');
         WHEN others THEN
dbms_output.put_line('Error!');  END;
```

ORACLE TRANSACTION

- Oracle PL/SQL transaction oriented language. Oracle transactions provide a data integrity. PL/SQL transaction is a series of SQL data manipulation statements that are work logical unit. Transaction is an atomic unit all changes either committed or rollback.
- At the end of the transaction that makes database changes, Oracle makes all the changes permanent save or may be undone. If your program fails in the middle of a transaction, Oracle detect the error and rollback the transaction and restoring the database.
- You can use the COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION command to control the transaction.
- COMMIT : COMMIT command to make changes permanent save to a database during the current transaction.
- ROLLBACK : ROLLBACK command execute at the end of current transaction and undo/undone any changes made since the begin transaction.
- SAVEPOINT : SAVEPOINT command save the current point with the unique name in the processing of a transaction.
- AUTOCOMMIT : Set AUTOCOMMIT ON to execute COMMIT Statement automatically.
- SET TRANSACTION : PL/SQL SET TRANSACTION command set the transaction properties such as read-write/read only access.

Commit

The COMMIT statement to make changes permanent save to a database during the current transaction and visible to other users,

```
SQL>COMMIT
BEGIN
    UPDATE emp_information SET emp_dept='Web Developer'
            WHERE emp_name='Saulin';
   COMMIT;
END;
/
```

Rollback

The ROLLBACK statement ends the current transaction and undoes any changes made during that transaction. If you make a mistake, such as deleting the wrong row from a table, a rollback restores the original data. If you cannot finish a transaction because an exception is raised or a SQL statement fails, a rollback lets you take corrective action and perhaps start over.

SQL>ROLLBACK [To SAVEPOINT_NAME];

```
SQL>DECLARE
emp_idemp.empno%TYPE;
BEGIN
  SAVEPOINT dup_found;
  UPDATE emp SET eno=1
            WHERE empname = 'Forbs ross'
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    ROLLBACK TO dup_found;
END;
/
```

Above example statement is exception raised because eno = 1 is already so DUP_ON_INDEX exception rise and rollback to the dup_foundsavepoint named.

Savepoint

- SAVEPOINT savepoint_namesmarks the current point in the processing of a transaction.

Savepoints let you rollback part of a transaction instead of the whole transaction.

```
SQL>SAVEPOINT SAVEPOINT_NAME;

SQL>DECLARE
emp_idemp.empno%TYPE;
BEGIN
  SAVEPOINT dup_found;
  UPDATE emp SET eno=1
            WHERE empname = 'Forbs ross'
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    ROLLBACK TO dup_found;
END;
/
```

Autocommit

- No need to execute COMMIT statement every time. You just set AUTOCOMMIT ON to execute COMMIT Statement automatically. It's automatic execute for each DML statement. set auto commit on using following statement,

```
SQL>SET AUTOCOMMIT ON;

SQL>SET AUTOCOMMIT OFF;
```

Set Transaction

- SET TRANSACTION statement is use to set transaction are read-only or both read write.

you can also assign transaction name.

```
SQL>SET TRANSACTION [ READ ONLY | READ WRITE ]
            [ NAME 'transaction_name' ];
```

Set transaction name using the SET TRANSACTION [...] NAME statement before you start the transaction.

SQL>SET TRANSACTION READ WRITE NAME 'tran_exp';

LOCK

- Types of Locks
- Level of Locks
- Explicit Locking
- Using Lock table statement
- Releasing locks
- Explicit lock using SQL & PL/SQL
- Deadlock

Lock [Concurrency Control]

- Locks are mechanisms used to ensure data integrity while allowing maximum concurrent access of data.
- Oracle locking is fully automatic & requires no user intervention.
- The oracle engine(server machine)locks table data while executing SQL stmt.This type of locking is called “implicit locking”.
- Oracle default locking strategy is implicit locking.
- Since the oracle engine has a fully automatic strategy ,it has to decide on two issues:- 1)Types of lock to be applied.

2)Level of lock to be applied.

- Types of Lock:-

Shared Locks

- Exclusive Locks

1)Shared Locks:-

a)Shared locks are placed on resource whenever a READ operation(select)is performed.

b)Multiple shared locks can be simultaneously set on a resource.

2)Exclusive Locks:-

a)Exclusive locks are placed on resource whenever WRITE operations (Insert, Update & Delete) are performed.

b)Only 1 exclusive lock can be placed on a resource at a time.

• Level Of Locks:-

A table can be decomposed into rows & a row can be further decomposed into fields.

1)Row Level

2)Page Level

3)Table Level

1)Row Level:-If the Where clause evaluates to only one row in the table.

2)Page Level:- If the Where clause evaluates to a set of data.

3)Table Level:- If there is no Where clause(i.e. the query accesses the entire table).

- Explicit Locking:-

The technique of lock taken on a table or its resources by a user is called "Explicit Locking".

Who can Explicitly Lock?

- Users can lock tables they own or any table on which they have been granted table privilege(select ,insert, update , delete)
- Table or rows can be explicitly locked by using either the select …for update stmt. Or Lock table stmt.
- The select …. For Update statement:-
- This clause is generally used to signal the oracle engine that data currently being used needs to be updated.

EXAMPLE

- Ex:-Two client machines client A & client B are recording the transaction performed in a bank for a particular account no. simultaneously.
- Client A fires the following select statement:
- Client A>select * from acct_mstr where acct_no='Sb9' for update;
- When the above select statement is fired the oracle engine locks the record 'sb9'. This lock is released when a commit or rollback is fired by client A Now client B fires a select stmt.,which points to record sb9

• Using Lock table statement:-

Purpose:-

- Use the LOCK TABLE statement to lock one or more tables, table partitions, or table sub partitions in a specified mode. This lock manually overrides automatic locking and permits or denies access to a table or view by other users for the duration of your operation.ALOCK is a mechanism that prevents destructive interaction between two simultaneous transactions or sessions trying to access the same database object.

Syntax:-

LOCK TABLE<TableName>[,<TableName>]…

IN{ROW SHARE|ROW EXCLUSIVE|SHARE UPDATE|

SHARE|SHARE ROW EXCLUSIVE|EXCLUSIVE}

[NOWAIT]

lock mode Clause

Specify one of the following modes:

- ROW SHARE ROW SHARE permits concurrent access to the locked table but prohibits users from locking the entire table for exclusive access. ROW SHARE is synonymous with SHARE UPDATE, which is included for compatibility with earlier versions of Oracle Database.
- ROW EXCLUSIVE ROW EXCLUSIVE is the same as ROW SHARE, but it also prohibits locking in SHARE mode. ROW EXCLUSIVE locks are automatically obtained when updating, inserting, or deleting.SHARE UPDATE See ROW SHARE.
- SHARE SHARE permits concurrent queries but prohibits updates to the locked table.
- SHARE ROW EXCLUSIVE SHARE ROW EXCLUSIVE is used to look at a whole table and to allow others to look at rows in the table but to prohibit others from locking the table in SHARE mode or from updating rows.
- EXCLUSIVE EXCLUSIVE permits queries on the locked table but prohibits any other activity on it.
- NOWAIT
- Specify NOWAIT if you want the database to return control to you immediately if the specified table, partition, or table sub partition is already locked by another user. In this case, the database returns a message indicating that the table, partition, or sub partition is already locked by another user.
- If you omit this clause, then the database waits until the table is available, locks it, and returns control to you.

- Example:-

The following statement locks the employees table in exclusive
mode but does not wait if another user already has locked the table:

```
LOCK TABLE employees IN EXCLUSIVE MODE NOWAIT;
```

Output:- Table
Locked.

- Releasing locks:-

All locks are released under the following circumstances:
1)The transaction is committed successfully.
2)A rollback is performed
3)A rollback to a savepoint will release locks set after the
specifiedsavepoint.
Note:- commit:-Save Work done.
Savepoint: Identify a point in a transaction to which you can later rollback.
Rollback: Restore database to original since the last COMMIT
GRANT/REVOKE: Grant or back permission to or from the oracle users.

- Deadlock:-
- A deadlock is a condition where two or more users are waiting for data locked by each other. Oracle automatically detects a deadlock and resolves them by rolling back one of the statements involved in the deadlock, thus releasing one set of data locked by that statement. Statement rolled back is usually the one which detects the deadlock.

Example:- Transaction1
BEGIN
UPDATE ACCT_MSTR SET CURBAL=500 WHERE ACCT_NO='SB1';
UPDATE ACCT_MSTR SET CURBAL=2500 WHERE ACCT_NO='CA2'; END
Transaction2:
BEGIN
UPDATE ACCT_MSTR SET CURBAL=5000 WHERE ACCT_NO='CA2';
UPDATE ACCT_MSTR SET CURBAL=3500 WHERE ACCT_NO='SB1';
END
Assume that TR1 & TR2 begin exactly at the same time.by default Oracle automatically places exclusive lock on data that is being updated.This causes TR1 to wait for TR2 to complete but in turn TR2 has to wait for TR1 to complete.

PL/SQL Procedures

- PL/SQL procedures create using CREATE PROCEDURE statement. The major difference between PL/SQL function or procedure, function return always value where as procedure may or may not return value.
- When you create a function or procedure, you have to define IN/OUT/INOUT parameters parameters.
- IN : IN parameter referring to the procedure or function and allow to overwritten the value of parameter. OUT : OUT parameter referring to the procedure or function and allow to overwritten the value of parameter.
- INOUT : Both IN OUT parameter referring to the procedure or function to pass both IN OUT parameter, modify/update by the function or procedure and also get returned.
- IN/OUT/INOUT parameters you define in procedure argument list that get returned back to a result. When you create the procedure default IN parameter is passed in argument list. It's means value is passed but not returned. Explicitly you have define OUT/IN OUT parameter in argument list.

```
CREATE [OR REPLACE] PROCEDURE [SCHEMA..] procedure_name
      [ (parameter [,parameter]) ]
   IS
      [declaration_section
                  variable declarations;
                  constant declarations;
      ]
  BEGIN
       [executable_section
                  PL/SQL execute/subprogram body
      ]
    [EXCEPTION]
         [exception_section
                  PL/SQL Exception block
```

```
    ]
 END [procedure_name];
/
```

PL/SQL Procedure Example

- In this example we are creating a procedure to pass employee number argument and get that employee information from table. We have emp1 table having employee information,

Create PROCEDURE

- In this example passing IN parameter (no) and inside procedure SELECT ... INTO statement to get the employee information.

```
CREATE or REPLACE PROCEDURE pro1(no in number,temp out emp1%rowtype)
IS
BEGIN
   SELECT * INTO temp FROM emp1 WHERE eno = no;
END;
/
```

PL/SQL Program to Calling Procedure

This program (pro) call the above define procedure with pass employee number and get that employee information.

```
SQL>
DECLARE
   temp emp1%rowtype;
   no number :=&no;
BEGIN
   pro1(no,temp);
dbms_output.put_line(temp.eno||'          '||
temp.ename||'                             '||
temp.edept||'   '||
            temp.esalary||' '||);
END;
/
```

PL/SQL Drop Procedure

You can drop PL/SQL procedure using DROP PROCEDURE statement,

```
DROP PROCEDURE procedure_name;
```

```
Drop procedure pro1;
```

PL/SQL Functions

- PL/SQL functions block create using CREATE FUNCTION statement. The major difference between PL/SQL function or procedure, function return always value where as procedure may or may not return value.
- When you create a function or procedure, you have to define IN/OUT/INOUT parameters parameters.
- IN : IN parameter referring to the procedure or function and allow to overwritten the value of parameter. OUT : OUT parameter referring to the procedure or function and allow to overwritten the value of parameter.
- IN OUT : Both IN OUT parameter referring to the procedure or function to pass both IN OUT parameter, modify/update by the function or procedure and also get returned.
- IN/OUT/INOUT parameters you define in function argument list that get returned back to a result. When you create the function default IN parameter is passed in argument list. It's means value is passed but not returned. Explicitly you have define OUT/IN OUT parameter in argument list.

PL/SQL Functions Syntax

```
CREATE [OR REPLACE] FUNCTION [SCHEMA..] function_name
      [ (parameter [,parameter]) ]
      RETURN return_datatype
   IS | AS
      [declaration_section
                variable declarations;
                constant declarations;
      ]
    BEGIN
       [executable_section
                PL/SQL execute/subprogram body
      ]
    [EXCEPTION]
         [exception_section
                PL/SQL Exception block
         ]
    END [function_name];
```

Function Example

In this example we are creating a function to pass employee number and get that employee name from table. We have emp1 table having employee information,

Solets start passing IN parameter (no). Return datatype set varchar2. Now inside function SELECT ... INTO statement to get the employee name.

```
SQL>CREATE or REPLACE FUNCTION fun1(no in number)
RETURN varchar2
IS
   name varchar2(20);
BEGIN
   select ename into name from emp1 where eno = no;
return name;
END;
/
```

PL/SQL Program to Calling Function

This program call the above define function with pass employee number and get that employee name.

```
SQL>DECLARE
          no number :=&no;
          name varchar2(20);
BEGIN
name := fun1(no);            dbms_output.put_line('Name:'||'
    '||name);
 end;
/
```

PL/SQL Drop Function

You can drop PL/SQL function using DROP FUNCTION statements.

```
DROPFUNCTIONfunction_name;
```

```
DROPFUNCTION fun1;
```

PL/SQL Packages

- PL/SQL Packages is schema object and collection of related data type (variables, constants), cursors, procedures, functions are defining within a single context. Package are device into two part,
- Package Specification
- Package Body
- Package specification block you can define variables, constants, exceptions and package body you can create procedure, function, subprogram.

- PL/SQL Package Advantages

- You can create package to store all related functions and procedures are grouped together into single unit called packages.
- Package are reliable to granting a privileges.
- All function and procedure within a package can share variable among them.
- Package are support overloading to overload functions and procedures.
- Package are improve the performance to loading the multiple object into memory at once, therefore, subsequent calls to related program doesn't required to calling physically I/O.
- Package are reduce the traffic because all block execute all at once.

PL/SQL Package Syntax

PL/SQL Specification :This contain the list of variables, constants, functions, procedure names which are the part of the package. PL/SQL specification are public declaration and visible to a program.

```
CREATE [OR REPLACE] PACKAGE package_name
   IS | AS
   [variable_declaration ...]
   [constant_declaration ...]
   [exception_declaration ...]
   [cursor_specification ...]
            [PROCEDURE [Schema..] procedure_name
                        [ (parameter {IN,OUT,IN OUT} datatype [,parameter]) ]
            ]
            [FUNCTION [Schema..] function_name
                        [ (parameter {IN,OUT,IN OUT} datatype [,parameter]) ]
                        RETURN return_datatype
            ]
END [package_name];
```

PL/SQL Body : This contains the actual PL/SQL statement code implementing the logics of functions, procedures which are you already before declare in "Package specification".

```
CREATE [OR REPLACE] PACKAGE BODY package_name
   IS | AS
   [private_variable_declaration ...]
   [private_constant_declaration ...]
            BEGIN
                        [initialization_statement]
                        [PROCEDURE [Schema..] procedure_name
                                    [ (parameter [,parameter]) ]
                                    IS | AS
                                                variable declarations;
                                    constant declarations;
                        BEGIN
```

```
                    statement(s);
            EXCEPTION
            WHEN ...
                    END
              ]
              [FUNCTION  [Schema..] function_name
                    [ (parameter [,parameter]) ]
                    RETURN return_datatype
                IS | AS
                    variable declarations;
                    constant declarations;
                BEGIN
                    statement(s);
            EXCEPTION
            WHEN ...
                    END
              ]
        [EXCEPTION
            WHEN built-in_exception_name_1 THEN
                User defined statement (action) will be taken;
        ]
END;
/
```

PL/SQL Package Example

PL/SQL Package example step by step explain to you, you are create 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	ZeniaSroll	Web Developer	42k

Package Specification Code

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 Code

Create Package body code for implementing procedure or function that are 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.

```
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;
/
```

PL/SQL Package Alter

You can update package code you just recompile the package body,

```
ALTER PACKAGE package_name COMPILE BODY;
```

Recompile the already created/executed package code,

```
SQL>ALTER PACKAGE pkg1 COMPILE BODY;
```

Package body Altered.

PL/SQL Package Drop

You can drop package using package DROP statement,

```
DROP PACKAGE package_name;
```

Drop the pkg1 program that was we created,

```
SQL>DROP PACKAGE pkg1;
```

Package dropped.

Overloading Package

Overloading :one procedure or function with the same name but with different paramenter can be define within package or within ol/sql block its called overloading

Function Overloading
Multiple function that are declared with the same name are called function overloading

Example

```
create or replace package tesst is
function f2(n1 in int,n2 in int,n3 in int)        return
int;
        function f2(n1 in int,n2 in int)
        return int;
end tesst;
/
```

```
create or replace package body tesst
is
function f2(n1 in int,n2 in int,n3 in int)return int      is
                tt int;    begin              tt:=n1+n2+n3;
                return tt;
          end f2;
           function f2(n1 in int,n2 in int)return int
is              t int;
       begin
       t:=n1+n2;
       return t;           end
f2; end tesst;
/

set serveroutput on declare
aint;           b
int;
begin
a:=tesst.f2(10,20,30);          b:=tesst.f2(10,20);
       dbms_output.put_line(a);
           dbms_output.put_line(b);
end;
```

Procedure Overloading

Multiple Procedure that are declared with the same name are called procedure overloading

```
create or replace package test is     procedure f1(n1 in
int,n2 in int,n3 in int,tt out int);     procedure f1(n1 in
int,n2 in int,t out int); end test;

create or replace package body test
is
   procedure f1(n1 in int,n2 in int,n3 in int,tt out int)
is      begin           tt:=n1+n2+n3;     end f1;
procedure f1(n1 in int,n2 in int,t out int)     is
   begin
t:=n1+n2;      end
f1;
end test;

set serveroutput on declare
aa  int;
```

```
bb int;
begin
test.f1(10,20,aa);        test.f1(10,20,30,bb);
dbms_output.put_line(aa);
dbms_output.put_line(bb); end;
```

PL/SQL Trigger

- Oracle engine allows the definition of procedures that are implicitly executed (executed by oracle engine),when an insert ,update or delete is issued against a table from application , these procedures are called database trigger
- They are fired implicitly.
- They are called by user.

Use of Database Trigger

- It can permit DML statements against a table only if they are issued , during regular business hours or on weekdays.
- It can also be used to keep an audit trail of table.
- It can be used to prevent invalid transactions.
- Enforce complex security authorizations.

Trigger V/s Procedure

- Trigger do not accept parameters where as procedure can.
- Trigger is executed implicitly by the oracle engine itself
- To execute a procedure , it has to be explicitly called by user

How To apply Database Trigger

- Trigger has three basic parts:

1)Trigger event or action
2)Trigger restriction
3)Trigger action.

1. Trigger event or action

•It is SQL statement that causes a trigger to be fired . it can be INSERT,UPDATE OR DELETE statement for specific table

2. Trigger restriction

- It must be specifiesboolean expression that must be TRUE for trigger to fire.
- It is an option available for trigger that are fired for each row.
- Its function is to conditionally control the execution of trigger.
- A trigger restriction is specified using when clause.

3. Trigger action.

•For row triggers ,the statement the PL/SQL block have access to column values(:new and :old) of the current row being processed.

Types Of Trigger

1. Row trigger
2. Statement trigger 3.Before trigger

4.After trigger.

1. Row trigger

- Row trigger is fired each time a ROW in the table is affected by the triggering statement.
- Row triggers should be used when some processing is required whenever a triggering statement affect a single row in a table.

2. Statement trigger

- Before trigger are use to derive specific column values before completing a triggering INSERT OR UPDATE statement.

3. Before trigger

- Before trigger are use to derive specific column values before completing a triggering INSERT OR UPDATE statement.

4. After trigger.

- AFTER trigger are used when triggering statement should complete before executing the trigger action.
- If BEFORE trigger is already present , an AFTER trigger can perform different action on same trigger statement.

Combination trigger

- BEFORE statement trigger.
  Before executing the triggering statement , the trigger action is executed.

- BEFORE row trigger.
  Before modifying each row affected by the triggering statement and before applying appropriate integrity constraints , the trigger is executed.

- AFTER statement trigger.
  after executing the triggering statement and applying any integrity constraints , the trigger action is executed.

- AFTER row trigger.
  - after modifying each row affected by the triggering statement and applying appropriate integrity constraints , the trigger is executed for current row.
  - Unlike BEFORE row triggers , AFTER row trigger have rows locked

Syntax

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

•\CREATE [OR REPLACE] TRIGGER trigger_name: Creates or replaces an existing trigger with the trigger_name.
•{BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
•{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 o NEW AS n]: 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.

Deleting Trigger

- Drop trigger <trigger name>
- Where trigger name is the name of the trigger to be dropped.

Example

```
create table custmast(custno number(3),name varchar2(20),address varchar2(20),dob date, occupation
varchar2(20));

insert into custmastvalues('101','AAA','Ring Road','13-jul-2000','analyst'); insert into
custmast values('102','BBB','Sumul dairy Road','13-sep-1999','programmer'); insert into
custmast values('103','CCC','Athwalines','02-jun-2001','banker'); insert into custmast
values('104','DDD','Adajan','22-jul-1995','manager');

create table audit_cust(custno number(3),name varchar2(20),address varchar2(20),dob date,
occupation varchar2(20),oper varchar2(10),operdate date);

CREATE OR REPLACE trigger audit_trail
AFTER update or delete
ON custmast

FOR EACH ROW

DECLARE
oper varchar2(10);
BEGIN       if updating
then
                    oper:='UPDATE';
          elsif deleting then
                    oper:='DELETE';
          end if;

          insert into audit_cust
values(:old.custno,:old.name,:old.address,:old.dob,:old.occupation,oper,sysdate);
END;

update custmast set occupation='marketing mgr' where custno=101; delete
from custmast where custno=102;
```

Trigger Using Primary key

1. Sequence

```
create table cust
(c_novarchar(20) primary key,cname varchar(20),city varchar(15));
insert into cust values(01,'chirag','surat'); insert
into cust values(02,'rahul','mumbai'); insert into
cust values(03,'bhavesh','delhi'); insert into cust
values(04,'pankaj','surat');
insert into cust values(05,'dharmesh','delhi');

create sequence c increment by 1 start with 1;

create or replace  trigger cust1
      before insert on cust
         for each row
declare
primary_key_valuevarchar(20); begin
      select 'c'||to_char(c.nextval)
         into primary_key_value
from dual;
         :new.c_no:=primary_key_value;
end;
/
```

2.MAX Function

```
Create table emp_mst191
(Emp_no int primary key,Emp_name varchar(20));

Create or replace trigger triempno
Before insert on emp_mst191
For each row
Declare
         Max_pkey_valuevarchar(30);
         New_pkey_valuevarchar(30);
Begin
```

```
 Select nvl(max(to_numbeer(substr(emp_no))),0) Into max_pkey_valueFrom emp_mst191;
          New_pkey_value := to_char(to_number(max_pkey_value)+1));
          New.emp_no : = 'E' || new_pkey_value;
End;
```

3.Look up Table

```
Create table branch_mst
(Branch_nonumber,Bname varchar(10));

Create  tablepkey_lookup
(pkey_value varchar2(20));

Create or replace trigger branch_no_genration
Before insert on branch_mst
For each row
Declare
Lookup_pkey_valuevarchar(20);
New_pkey_valuevarchar(20);
Begin
      Select pkey_value into lookup_pkey_value from pkey_lookup;
      Exception
      When no_data_found then lookup_pkey_value:='b1';
      End;
:new.branch_no:=lookup_pkey_value;
New_pkey_value:=to_number(substr(lookup_pkey_value,2,1))+1;
Lookup_pkey_value:='b'|| new_pkey_value;
        If lookup_pkey_value='b2' then
  Insert into pkey_lookup values(lookup_pkey_value);
         Else
              Update pkey_lookup set pkey_value=lookup_pkey_value;
        End if;
End;
/
```

Raise Application Error

```
CREATE OR REPLACE trigger chk_validqty
BEFORE insert
ON item
FOR EACH ROW
```

```
BEGIN
        IF :new.qty< 5 then
  RAISE_APPLICATION_ERROR(-20001,'cant insert data because qty<5');  end if;
END;

insert into item values(100,'pen','red',1,10)
```