Unit 4

Introduction to PL/SQL :-

★ SQL is the natural language of DBA, but it suffers from various disadvantages.

1. SQL doesn't have any procedural capabilities.
2. SQL statements are passed to oracle engine one at a time.

★ Oracle provides PL/SQL as a fully structure programming language.

Advantage of PL/SQL :-

① PL/SQL is a development tool that not only support SQL data manipulation but also facilities.
- → conditional checking
- → branching
- → looping

② It sends an entire block of SQL statement to oracle engine all in one go.

③ It deals with error as required.

④ It allow declaration and use of variable in block of code.

⑤ Application written in PL/SQL are portable to any

computer
→ hardware
→ operating system.

PL/SQL Block:-

★ A PL/SQL has a definite structure which can be divided into following sections.

1. Declare section
2. Begin section
3. Exception section
4. End section.

Declare — Declarations of memory variable, contants cursors etc in PL/SQL.

Begin — SQL executable statements PL/SQL executable statement.

Exceptions — SQL or PL/SQL code to handle errors that may arise execution of the code block between Begin and exception section.

END;

Fig:- PL/SQL Block structure

PL/SQL in oracle Engine

★ The PL/SQL engine resides in oracle engine, the oracle engine can process not only single SQL statement but also entire PL/SQL blocks.

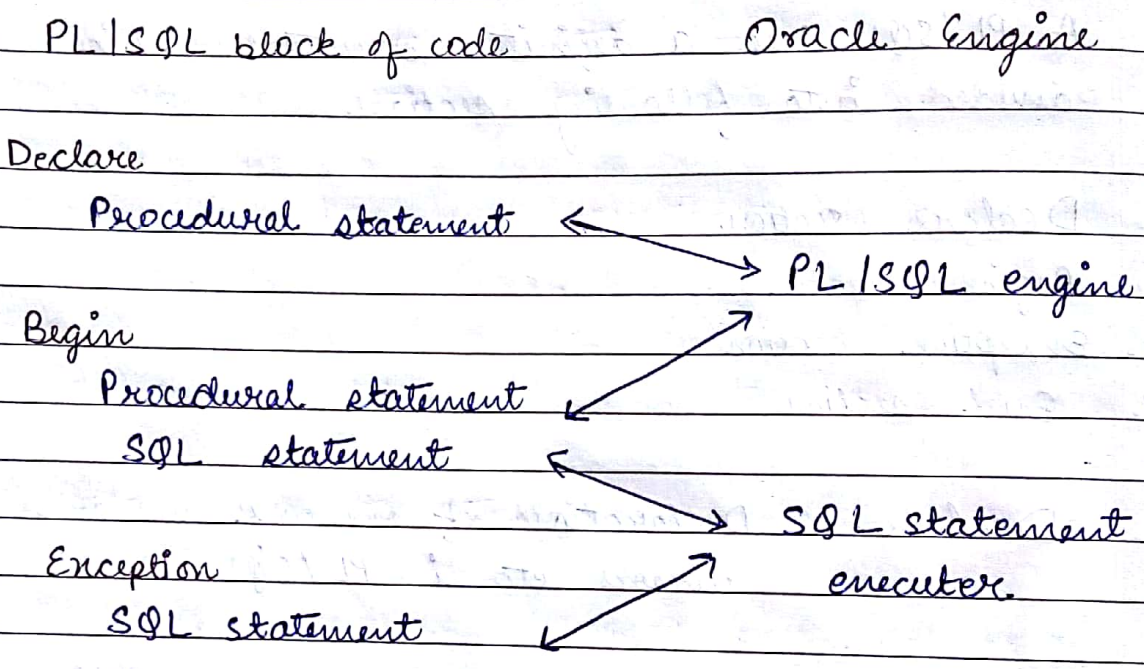

Fig:- PL/SQL Execution environment.

1. Character Set :-
   - → Upper case (A-Z)
   - → Lower case (a-z)
   - → Numericals (0-9)
   - → Symbols

Lexical unit :- Words used in PL/SQL block are called Lexical unit.

2. Literals :- A literal is a numeric value or a character string used to represent it set.

(a) Numerical Literal :- → integer
→ float

(b) String Literal :- 'Hello World'

(c) Character Literal :- 'A'

(d) Logical Literal :- → true
→ false
→ Null

PL/SQL Data types :-
→ Number
→ Boolean
→ char
→ date

★ % type attribute provide for further integration PL/SQL use % TYPE attribute to declare variables based on the definitions of column in a table.

★ Not NULL causes creations of a variables or a constant that can't be assigned or null value.

Variables :- A variable name must begin with a character and can be followed by a max of 29 other character.

Assigning values to variable:-
There are 2 method :→

(a) using assignment operator :=
(b) selecting or fetching that data values into variables.

Constant :- Keyword constant must be added to variables name and a value assigned immediately.

RAW :- Raw types are used to store binary data.

LOB :- used to store large object.

Large object can be either → binary value } 4 GB in size
→ character value }

LOB (types)

BLOB (Binary LOB)
(Stores unstructured binary data upto 4 G.B) of video or picture.

CLOB (character LOB)
store single byte character upto 4 G.B store documents

BFILE (Binary File)
This stores pointer binary data stored as an external file outside the database.

Logical comparision :- → Relational (<, >, <=, >=, <>)
→ Logical operator AND, OR, NET

Displaying user message or VDU (visual display unit) screen:-

1. To display message, SERVER OUTPUT should be to ON.

2. Put_line function puts a piece of information in the package buffer followed by end_of_line marks.
Syntax:- SET SERVEROUTPUT [ON/OFF]

Comment :- ① double hyphen (--)
② /* — */ for multiple lines.

I^st program in PL/SQL :-

```
Set serveroutput on;
Declare
    message varchar2 [20] := 'Hello World';
Begin
    dbms_output.put_line (message);
```

3. dbms_output is a package that includes a number of procedures and function that accumulate info. in buffer so that it can be retrived later.

serveroutput is a SQL PLUS enviroment primeter that display the information pass as a parameter to the put line function.

Control Structure :-
→ conditional statements
→ Iterative control
→ Sequential control.

① Conditional Statement :-

(a) IF_THEN STATEMENT :-

Syntax :-

```
IF condition
  THEN
  // Block of statement
  END IF;
```

(b) IF_THEN_ELSE STATEMENT :-

Syntax :-

```
IF condition
  THEN
  // Block of statement
ELSE
  // Block of statement
END IF;
```

(c) IF_THEN_ELSE IF STATEMENT :-

Syntax :-

```
IF condition 1
THEN
  Block of statement
ELSE IF condition 2
  Block of statement
ELSE
  Block of statement
END IF;
```

Program :-

```
Set serveroutput on;
Declare
    var number (3) := 50;

Begin
    If (var = 10) THEN
    dbms_output.put_line ('value of var is 10');

    Else if (var = 20) THEN
    dbms_output.put_line ('value of var is 20');

    Else if (var = 30) THEN
    dbms_output.put_line ('value of var is 30');

    ELSE
        dbms_output.put_line ('Name of above
                               condition is true');

    END IF;
        dbms_output.put_line ('Exact value of
                               var is:' || var);

    END;
```

Switch Statement :-

Syntax :-

```
    Case [expression]
when condition 1 The block of statement 1.
when condition 2 The block of statement 2.
.....
.....
.....
```

when condition n then block of statement n
Else
Block of statement
END;

PL/SQL case statement example :→

```
Declare
    namechar char (1) := 'J';

Begin
    case namechar
    when 'B' then
    dbms_output.put_line ('Bharat');

    when 'R' then
    dbms_output.put_line ('Ruchi');

    when 'S' then
    dbms_output.put_line ('Sahdev');

    when 'V' then
    dbms_output.put_line ('Vinod');

    when 'J' then
    dbms_output.put_line ('July');

    else
    dbms_output.put_line ('No search name');
    end case;

END;
```

PL/SQL Exit loop :-

★ The PL/SQL loop repeatly executes a block of statements until it reaches a loop exit. Then exit and when exit are used to terminate a loop.

① Exit :- It is used to terminate loop unconditionally

② Exit when :- Used to terminate loop conditionally.

PL/SQL loop statement :-

Syntax :-

```
loop
  // block of statement
exit ;
end loop;
```

PL/SQL with Exit when :-

Syntax :-

```
loop
// block of statement
exit when condition ;
end loop;
```

Program :-

①

```
Declare
    num number := 1 ;

Begin
    loop
    dbms_output.put_line (num);
```

```
if num = 10
  exit ;
END if ;
  num := num + 1 ;
  end loop ;
END;
```

(2)

```
Declare
    num Number := 1 ;

Begin
      loop
        dbms_output.put_line (num);
        Exit when num = 10 ;
        num := num + 1 ;
        end loop;

END;
```

PL/SQL while loop :-

Syntax :-

```
while condition
    loop
        -- block of statements ;
    end loop ;
```

Example :-

```
Declare
      num number := 1 ;
```

```
Begin
    while num <= 10;
    loop
        dbms_output.put_line (num);
        num := num + 1;
    end loop;
END;
```

PL/SQL for in loop:-

Syntax:- For loop_counter in [reverse] start value .. end_value
loop
    -- block of statements;
end loop;

(i) The double dot specify the range operator.
(ii) By default the loop will starts from start_value ... end_value but we can reverse the loop process by using reverse.

example:-

```
Declare
    var number;

Begin
    for var in 1 .. 10
    loop
    dbms_output.put_line (var);
    end loop
END;
```

Output :- 1 to 10.

eg:-

```
Declare

Begin
    for var in reverse 1 ... 10
    loop
      dbms_output.put_line (var);
    end loop;
END;
```

output :- 10 to 9

(iii) The counter variable is incremented by one and does not incrementing explicitly.

Assigning SQL Query Result to PL/SQL variables:-

Example:- 1. Create table customers (
ID int not null,
Name varchar (20) not null,
Age Int not null,
Address char (25);
Salary Decimal (18, 1);
Primary Key (ID));

∴ Table created

2. Insert values into table
Insert into customers (ID, name, Age, Address salary)
Values (1, 'Ramesh', 32, 'Ahmedabad', 2000.00)

Insert into customers (ID, name, Age, Address, salary)
values (2, 'Khitan', 25, 'Delhi', 1500.00);

Insert into customers (ID, name, Age, Address, salary)
values (3, 'Kaushik', 23, 'Kota', 2000.00);

3. Declare
    c_id customers78.ID % type := 1;
    c_name customers78.Name % type;
    c_addr customers78.address % type;
    c_sal customers78.salary % type;

Begin
    Select name, address, salary into
    c_name, c_addr, c_sal
    from customers
    where id = c_id
    dbms_output.put_line ('customer' || c_name || 'from' ||
                          'earn' || c_sal);

END;

Write a PL/SQL code to increase the salary of those employee whose salary is less than 2000 or equal to 2000.

• Declare
    c_id customer.id % type := 1;
    c_sal customer.salary % type;

```
Begin
    Select Salary
    into c_sal
    from customers
    where id = c_id
    if (c_sal <= 2000) THEN
      Update customer
      set Salary := salary + 1000
       where id = c_id;
       dbms_output.put_line ('salary updated');
    end if;

END;
```

Write a PL/SQL code to calculate the area of a circle for a value of radius varying from 3 to 7. Now to store the radius and corresponding value of calculated area in an empty table named area consisting of two colouns radius and area.

```
• Create table Areas (Radius Number(5),
                      area Number(14,2));

Declare
    pi constant Number(4,2) := 3.14;
    radius number(5)
    area number(14,2);

Begin
    radius := 3;
    while radius <= 7
```

```
loop
    area := pi * power (radius, 2)
Insert into areas value (radius, area);
        radius = radius + 1;
 end loop;

END;
```

—X—

Processing a PL/SQL Block
→ Batch processing
→ Real time processing.

Cursors :- A technique that Oracle provides for manipulating table data in batch processing mode.

Oracle and processing of SQL statement :-

1. Reserve a private SQL area in memory.
2. Populates it user data requested in SQL.
3. Processes the data here
4. Free up memory area when task complete.

An SQL statement that display employee number (Emp_No), employee name (Ename) & department (DEM) from EMP_MSTR table in ascending order of Ename.

```
Select EMP_No, ENAME, DEPT from EMP_MSTR order by
                                        ENAME;
```

★ Oracle engine uses a work area for processing in order to execute SQL statement (cursor).

★ The data that is stored in the cursor is called the Active data set.

★ Two types of cursor
→ Implicit Cursor
→ Explicit cursor.

Creneral Cursor Attribute :-

Attribute Name	Description
% ISOPEN	Return true if cursor is open, FALSE otherwise.
% FOUND	Return TRUE if record was fetched successfully, FALSE otherwise.
% NOT FOUND	Return TRUE if record was not fetched successfully, FALSE otherwise.
% ROW COUNT	Return number of records processed from the cursor.

Implicit Cursor :-
They are automatically created by Oracle whenever an SQL statement is executed. Programmers can't

control them.

Using customer table we have to update the table (increase salary of each customer by 500) & use the SQL % ROW count attribute to determine the number of row affected.

- Select * from customer

ID	Name	Age	Address	Salary
1	Nidhi	22	Ahmedabad	2000
2	Shefu	24	Jodhpur	1500
3	Khush	23	Jaipur	3000
4	Sarabjeet	25	Delhi	4500

```
Declare
    total_row number(2);
Begin
    Update customers
        Set Salary = Salary + 500;
    if SQL % not found then
        dbms_output.put_line ('No customer selected');
    else if SQL % found then
        total rows := SQL % rowcount;
        dbms_output.put_line (total_rows || 'customer
                                selected');
    end if;
END;
```

Explicit cusor :-

* These are programmer - defined cursor for gaining more control over the control area.

* It should be defined in the declaration section of PL/SQL block.

* It is created on a select statement which return more than one row.

Use explicit cursor :-

1. Declaring the cursor for initializing the memory.
2. Opening the cursor for allocation of memory.
3. Fetching the cursor for retriving the data.
4. closing the cursor cursor to release allocated memory.

eg:- Declare

```
c_id customer.id %. type ;
c_name customer.name %. type ;
c_add customer.address %. type ;
cursor c_customers is
SELECT id, name, address from customers ;
Begin
Open c_customers ;
```

```
loop
Fetch C_customer into c_id, c_name, c_add;
EXIT WHEN C_customer %notfound;
dbms_output.put_line (c_id || ' ' || C_name || ' '
                      || c_add);
End loop;
Close C_customers;

END;
```

Exceptional Handling :-

★ An exception is an error condition during a program execution. PL/SQL supports programmers to catch such conditions using exception block in program.

Pre-defined Exceptions :-

① DUP_VAL_ON_INDE :- Duplicate values are attempted to store in a column with unique index.

② INVALID_NUMBER :- Conversion ~~failure~~ of characters into a no. fails.

③ LOGIN_DENIED :- Log on DB with invalid
→ Username
→ Password.

④ No_Data_FOUND :- Return no rows.

(5) NOT_LOGGED_ON :- Database call without being on.

(6) PROGRAM_ERROR :- Internal problems.

(7) STORAGE_ERROR :- Ran out of memory

(8) ~~Arithmatics, size~~

(8) VALUE_ERROR :- Arithmatic, size, transaction error

(9) ZERO_DIVIDE :- Divide a no. by zero.

(10) OTHERS :- All other exception

eg:-

```
Declare
    c_id customers.id %type := 8;
    c_name customers.name %type;
    c_add customers.address %type;

Begin
    SELECT name, address INTO c_name, c_add.
    FROM customer
    WHERE id = c_id;
    dbms_output.put_line ('Name' || c_name);
    dbms_output.put_line ('Address' || c_add);

Exception
    WHEN no_data_found THEN
    dbms_output.put_line ('No Such customers');
    WHEN other THEN
    dbms_outpat.put_line ('Error');

END;
```

Triggers:- Triggers are stored program, which are automatically executed or fired. when some event occurs.

→ DML (delete, update, insert)
→ DDL (create, alter, Drop)
→ Database operation (Server error, logon, log off startup and shutdown).

Trigger creations:-

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 ON as n ]
[For each ROW]
 WHEN (condition)
 DECLARE
     Declaration_statement
 BEGIN
     Executable_statement
 Exception
     Exception
 END;
```

eg:-

```
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;
```

Output :- Trigger created;

```
Insert into customer (ID, Name, Age, Address, Salary);
Value ('7', 'Kirti', 22, 'HP', 7500.00);
```

output :- Old salary
New salary : 7500
salary difference;

```
Update customers
Set salary = salary + 500
where id = 2;
```

Old salary : 1500
New salary : 2000
Salary difference : 500

X

Procedures :- PL/SQL provides 2 types of subprogram
(a) function
(b) Procedure → These sub program don't return a value directly and mainly used to program an action.

Syntax :-

```
Create [or Replace] PROCEDURE procedure_name
[(parameter_name [IN | OUT | INOUT] type [...])]
{is | AS}
BEGIN
    < procedure_body >
END procedure_name;
```

IN :- Represents value that will be passed from outside.

OUT :- Represents parameter that will be used to return a value.

IN/OUT :- It passes an initial value to a subprogram and returns a updated value to the caller.

eg :-
```
create or Procedure Greetings
As
Begin
  dbms_output.put_line ("Hello World");
END;
```

Executing a standalone procedure :-

(a) Using Execute keyword → EXECUTE greetings.
output :- Hello World.

(b) Calling the name of the procedure from PL/SQL block.

→
```
Begin
  greetings;
END;
```
output :- Hello World.

Deleting a procedure :-

syntax :-

```
DROP PROCEDURE procedure_name;
```

eg :-

```
DROP PROCEDURE greetings;
```

Packages :- It are schema objects that groups logically related PL/SQL types variables and subprograms.

A package has 2 parts :- ① Package specification
② Package body / defination.

① Package specification :- It declare types, variables, constants exception, subprogram that can be refused becom outside the package.

② Package body / defination :- Private object any subprogram coded in package body is a private object.

Syntax :-

```
create Package cust_sal As
    procedure find_sal ()
END cust_sal;
```

output :- Package created.

eg :-

```
BEGIN
    cust_sal find_sal ();
END;
```