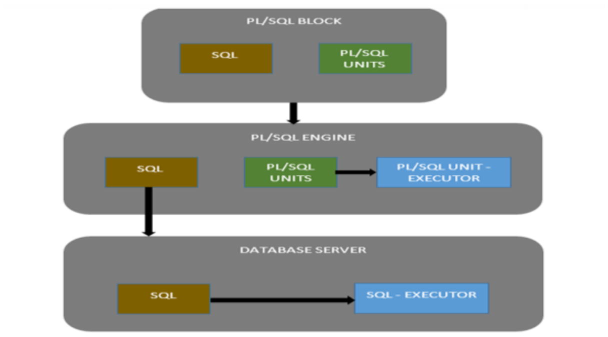
What is Oracle PL/SQL?

ORACLE PL/SQL is an extension of SQL language that combines the data manipulation power of SQL with the processing power of procedural language to create super powerful SQL queries. PL/SQL means instructing the compiler 'what to do' through SQL and 'how to do' through its procedural way.

Architecture of PL/SQL

The PL/SQL architecture mainly consists of following three components:

1. PL/SQL block
2. PL/SQL Engine
3. Database Server



Advantage of Using PL/SQL

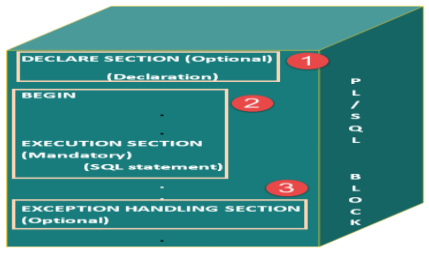
1. Better performance, as SQL is executed in bulk rather than a single statement
2. High Productivity
3. Tight integration with SQL
4. Full Portability
5. Tight Security
6. Support Object Oriented Programming concepts.

Block Structure

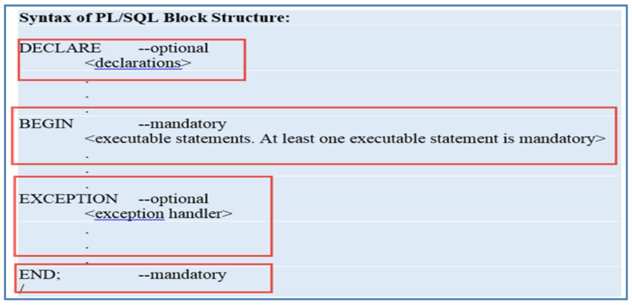
PL/SQL blocks have a pre-defined structure in which the code is to be grouped. Below are different sections of PL/SQL blocks.

1. Declaration section
2. Execution section
3. Exception-Handling section

The below picture illustrates the different PL/SQL block and their section order.



PL/SQL Block Syntax



```
Syntax of PL/SQL Block Structure:
DECLARE        --optional
        <declarations>
        .
        .
        .
BEGIN          --mandatory
        <executable statements. At least one executable statement is mandatory>
        .
        .
        .
EXCEPTION      --optional
        <exception handler>
        .
        .
        .
END;           --mandatory
/
```

Note: We need to execute "set serveroutput on" if we need to see the output of the code

PL/SQL has the following features:

- • PL/SQL is tightly integrated with SQL.
- • It offers extensive error checking.
- • It offers numerous data types.
- • It offers a variety of programming structures.
- • It supports structured programming through functions and procedures.
- • It supports object-oriented programming.
- • It supports the development of web applications and server pages.

Data types:

- CHARACTER Data Type
  - CHAR Data type (fixed string size)
  - VARCHAR2 Data type (variable string size)
  - VARCHAR Data type
  - NCHAR (native fixed string size)
  - NVARCHAR2 (native variable string size)
  - LONG and LONG RAW
- NUMBER Data Type
  - NUMBER(prec, scale): NUMBER(8,2); NUMBER(8); NUMBER;
  - FLOAT (floating-point type)
  - INT (Integer type)
- DATE Data Type
  - newyear DATE:='01-JAN-2015'; current_date DATE:=SYSDATE;

Properties of Identifiers

- Must start with a letter
- Maximum size is limited to 30 letters
- Cannot contain whitespace characters
- Can contain dollar sign ('$'), underscore ('_') and hash sign ('#')

- Is case-insensitive

  Example:

```
DECLARE

a integer := 10;

b integer := 20;

c integer;

f number;

BEGIN

c := a + b;

dbms_output.put_line('Value of c: ' || c);

f := 70.0/3.0;

dbms_output.put_line('Value of f: ' || f);

END;

/
```

Types of Decision Making Statements:

Oracle provides the following types of decision making statements.

- IF-THEN
- IF-THEN-ELSE
- IF-THEN-ELSIF
- NESTED-IF
- CASE
- SEARCHED CASE

```
DECLARE

mark NUMBER :=25;

BEGIN

        dbms_output.put_line('Program started.' );

        IF( mark >= 70) THEN

                dbms_output.put_line('Grade A');

        ELSIF(mark >= 40 AND mark < 70) THEN

                dbms_output.put_line('Grade B');

        ELSIF(mark >=35 AND mark < 40) THEN

                dbms_output.put_line('Grade C');

        ELSE

        dbms_output.put_line('No Grade');

        END IF;

        dbms_output.put_line('Program completed.' );

END;

/
```

Example(**Nested- If Statement):**

```
DECLARE

mark NUMBER :=25;

BEGIN

          dbms_output.put_line('Program started.' );

          IF( mark >= 70) THEN

                    dbms_output.put_line('Grade A');

          ELSIF(mark >= 40 AND mark < 70) THEN

                    dbms_output.put_line('Grade B');

          ELSIF(mark >=35 AND mark < 40) THEN

                    dbms_output.put_line('Grade C');

          ELSE

          dbms_output.put_line('No Grade');

          END IF;

          dbms_output.put_line('Program completed.' );

END;

/
```

Example (Case Statement):

```
DECLARE

 a NUMBER :=55;

b NUMBER :=5;

arth_operation VARCHAR2(20) :='DIVIDE';

BEGIN

dbms_output.put_line('Program started.' );

CASE

          WHEN arth_operation = 'ADD' THEN dbms_output.put_line('Addition of the numbers are: '||a+b );

          WHEN arth_operation = 'SUBTRACT' THEN dbms_output.put_line('Subtraction of the numbers are: '|| a-b);

          WHEN arth_operation = 'MULTIPLY' THEN dbms_output.put_line('Multiplication of the numbers are: '|| a*b );

          WHEN arth_operation = 'DIVIDE' THEN dbms_output.put_line('Division of the numbers are: '|| a/b );

          ELSE dbms_output.put_line('No operation action defined. Invalid operation');

END CASE;

          dbms_output.put_line('Program completed.' );

END;

/
```

Types of Loop in PL/SQL

PL/SQL provides following three types of loops

- Basic loop statement
- For loop statement
- While loop statement

Example (Loop):

```
DECLARE

        a NUMBER:=1;

BEGIN

        dbms_output.put_line('Program started.');

LOOP

        dbms_output.put_line(a);

        a:=a+1;

        EXIT WHEN a>5;

END LOOP;

        dbms_output.put_line('Program completed');

END;

/
```

Example (Loop with label):

```
DECLARE

        a NUMBER:=0;

        b NUMBER;

        upper_limit NUMBER :=4;

BEGIN

        dbms_output.put_line('Program started.' );

<<outer_loop>>

LOOP

        a:=a+1;

        b:=1;

        <<inner_loop>>

        LOOP

                EXIT outer_loop WHEN a > upper_limit;

                dbms_output.put_line(a);

                b:=b+1;

                EXIT inner_loop WHEN b>a;

        END LOOP;

END LOOP;

        dbms_output.put_line('Program completed.');
```

Example (For Loop & While Loop):

```
DECLARE

B NUMBER;

BEGIN

dbms_output.put_line('Program started' );

FOR  A IN 1..3

LOOP

        B:=1;

        WHILE (A>=B )

                LOOP

                        dbms_output.put_line(A);

                        B:=B+1;

                END LOOP;

END LOOP;

dbms_output.put_line('Program completed' );

END;

/
```

What is CURSOR in PL/SQL?

Oracle creates context area for processing an SQL statement which contains all information about the statement. A Cursor is a pointer to this context area.

Cursor Attribute	**Description**
%FOUND	It returns the Boolean result 'TRUE' if the most recent fetch operation fetched a record successfully, else it will return FALSE.
%NOTFOUND	This works oppositely to %FOUND it will return 'TRUE' if the most recent fetch operation could not able to fetch any record.
%ISOPEN	It returns Boolean result 'TRUE' if the given cursor is already opened, else it returns 'FALSE'
%ROWCOUNT	It returns the numerical value. It gives the actual count of records that got affected by the DML activity.

Example (Implicit Cursor):

```
DECLARE

        total_rows number(2);

BEGIN

        UPDATE emp

                SET sal = sal + 500 where job = 'MANAGER1';

        IF sql%notfound THEN

                dbms_output.put_line('no customers selected');

        ELSIF sql%found THEN

                total_rows := sql%rowcount;

        dbms_output.put_line( total_rows || ' customers selected ');

        END IF;

END;

/
```

Example (Explicit Cursor):

```
DECLARE

        c_Input_Eno emp.empno%type := &eno;

        c_ENO emp.empno%type;

        c_ENAME emp.ENAME%type;

        c_SAL emp.SAL%type;

        CURSOR c_EMP is SELECT EMPNO, ENAME, SAL FROM EMP;

        -- user defined exception

        ex_invalid_eno  EXCEPTION;

BEGIN

        OPEN c_EMP;

        IF c_Input_Eno  < 0 THEN RAISE ex_invalid_eno;

        END IF;

        LOOP

                FETCH c_EMP into c_ENO, c_ENAME, c_SAL;

                EXIT WHEN c_EMP%notfound;

                dbms_output.put_line(c_ENO || ' ' || c_ENAME || ' ' || c_SAL);

        END LOOP;

        CLOSE c_EMP;

        --Check default expectation

        SELECT empno, ename INTO c_ENO , c_ENAME  FROM emp WHERE empno= c_Input_Eno ;

EXCEPTION

        WHEN no_data_found THEN dbms_output.put_line('No such customer!'); --System defined

        WHEN ex_invalid_eno THEN dbms_output.put_line('ID must be greater than zero!'); --User Defined

END;

/
```

Example (Procedure):

```
CREATE OR REPLACE PROCEDURE pro_update (p_empno IN INT, p_amount IN FLOAT )

IS

        No_Record_Updated  EXCEPTION;

BEGIN

        UPDATE emp SET sal = sal + p_amount where empno = p_empno;

        IF sql%rowcount = 0 THEN

                RAISE No_Record_Updated;

        ELSE

                commit;

        END IF;

        dbms_output.put_line(fun_update(p_empno));

EXCEPTION

        WHEN No_Record_Updated THEN dbms_output.put_line('ID is not available in the emp table!');

END;

/
```

Example (Function):

```
create or replace FUNCTION fun_update (p_empno IN INT)

RETURN varchar2 IS

c_ENO emp.empno%type;

c_ENAME emp.ENAME%type;

c_SAL emp.SAL%type;

BEGIN

SELECT empno, ename, sal into c_ENO,c_ENAME,c_SAL from emp where empno = p_empno;

dbms_output.put_line(c_ENO || ' ' || c_ENAME || ' ' || c_SAL);

RETURN 'Function executed successfully'

END;

/
```

Create a Tracking Table:

```
/*Create following table which will be used in the function */

CREATE TABLE track_updates(

   EMPNO    NUMBER(4),

   ENAME    VARCHAR2(10),

   OLD_SAL  NUMBER(7,2),

   NEW_SAL  NUMBER(7,2)

);
```

Example (Trigger):

```
CREATE OR REPLACE TRIGGER tri_update

BEFORE DELETE OR INSERT OR UPDATE ON emp

FOR EACH ROW

DECLARE

                sal_diff number;

BEGIN

                sal_diff := :NEW.sal - :OLD.sal;

                dbms_output.put_line('EMPNO: ' || :OLD.empno);

                dbms_output.put_line('ENAME: ' || :OLD.ename);

                dbms_output.put_line('Old salary: ' || :OLD.sal);

                dbms_output.put_line('New salary: ' || :NEW.sal);

                dbms_output.put_line('Salary difference: ' || sal_diff);

--Enter values in the tracking table

INSERT INTO track_updates(empno, ename, old_sal, new_sal) VALUES ( :OLD.empno,:OLD.ename,
:OLD.sal, :NEW.sal );

END;

/
```

```
SQL> set serveroutput on

SQL> exec pro_update(7566, 700)

EMPNO: 7566

ENAME: JONES

Old salary: 3975

New salary: 4675

Salary difference: 700

Empno:7566Old Salary:  3975New salary:  4675

7566 JONES 4675

Function executed successfully

PL/SQL procedure successfully completed.
```

```
create or replace FUNCTION TDS (p_empno IN INT)

RETURN FLOAT IS

c_ENO emp.empno%type;

c_ENAME emp.ENAME%type;

c_SAL emp.SAL%type;

BEGIN

SELECT empno, ename, sal into c_ENO,c_ENAME,c_SAL from emp where empno = p_empno;

IF( c_SAL > 5000) THEN

RETURN c_SAL*0.30;

ELSE

RETURN c_SAL*0.20;

END IF;

END;

/
```

```
Select EMPNO, SAL, TDS(EMPNO) as Tax_Deducted_at_Source from emp;
```