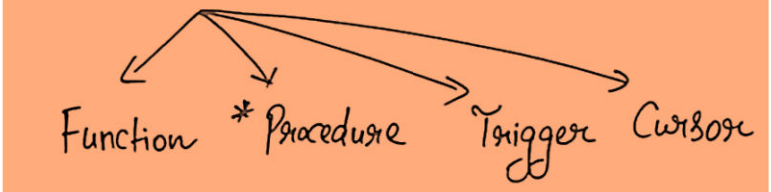
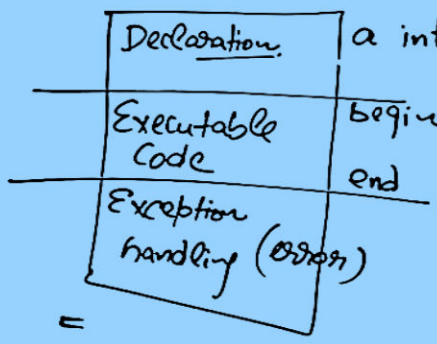
PL/SQL

PL-SQL (Procedural - SQL)



Block.
Code.



=

```
declare
a int
b int
c int
begin
a:=10;
b:=20
c:=a+b;
end.
```

SQL (write a query)
↓
Declarative.
'What to do'
How to do ✗

{

}

write a simple program for printing “Hello World”

```
SQL> connect
Enter user-name: hr
Enter password:
Connected.
SQL> set serveroutput on
SQL> begin
  2  dbms_output.put_line('hello world');
  3  end;
  4  /
hello world

PL/SQL procedure successfully completed.
```

Declaring and usage of variables in the program

```
  1  declare
  2  X varchar2(25);
  3  begin
  4  X:='Hello World';
  5  dbms_output.put_line (X);
  6* End;
SQL> /
Hello World

PL/SQL procedure successfully completed.
```

```
SQL> declare
  2  var1 varchar2(10);
  3  num1 number(3);
  4  begin
  5  var1 := 'Tutorials';
  6  num1 := 100;
  7  dbms_output.put_line('Val1 : ' || var1);
  8  dbms_output.put_line('Num1 : ' || num1);
  9  end;
 10  /
Val1 : Tutorials
Num1 : 100

PL/SQL procedure successfully completed.
```

```
SQL> declare
  2  name varchar2(10);
  3  sal number(10,2);
  4  begin
  5  select First_Name, Salary into name, sal from Employees
  6  where Employee_id = 100;
  7  dbms_output.put_line('Name : ' || name);
  8  dbms_output.put_line('Salary : ' || sal);
  9  end;
 10  /
Name : Steven
Salary : 24000

PL/SQL procedure successfully completed.
```

```
  1  declare
  2  name Employees.First_Name%TYPE;
  3  sal Employees.Salary%TYPE;
  4  lastname name%TYPE;
  5  begin
  6  select First_Name, Salary into name, sal from Employees
  7  where Employee_id = 100;
  8  dbms_output.put_line('Name : ' || name);
  9  dbms_output.put_line('Salary : ' || sal);
 10* end;
SQL> /
Name : Steven
Salary : 24000

PL/SQL procedure successfully completed.
```

If we don’t know the size of the column as exists in the table, employees then we can use the % Type for the declaration of the data type of the attributes in the PL/ SQL block.

```
SQL> declare
  2  record Employees%ROWTYPE;
  3  begin
  4  select * into record from Employees
  5  where Employee_Id = 100;
  6  dbms_output.put_line(record.First_Name || ' | ' || record.Last_Name || ' | ' || record.salary);
  7  end;
  8  /
Steven | King | 24000

PL/SQL procedure successfully completed.
```

PL/SQL - Conditional Statements

```
SQL> declare
  2     deptid number(2);
  3     sal number(10, 2);
  4  begin
  5     select Salary, Department_id into sal, deptid from Employees where Employee_Id = 105;
  6     dbms_output.put_line(sal || ' : ' || deptid);
  7     if deptid = 30 then
  8             sal := sal + 3;
  9     end if;
 10     dbms_output.put_line(sal || ' : ' || deptid);
 11  end;
 12  /
4800 : 60
4800 : 60

PL/SQL procedure successfully completed.

SQL>
```

```
 1  declare
 2     deptid number(2);
 3     sal number(10, 2);
 4  begin
 5     select Salary, Department_id into sal, deptid from Employees where Employee_Id = 105;
 6     dbms_output.put_line(sal || ' : ' || deptid);
 7     if deptid = 30 then
 8             sal := sal + 3;
 9     else
10             sal := sal + 10;
11     end if;
12     dbms_output.put_line(sal || ' : ' || deptid);
13* end;
SQL> /
4800 : 60
4810 : 60

PL/SQL procedure successfully completed.
```

```
 1  declare
 2     deptid number(2);
 3     sal number(10, 2);
 4  begin
 5     select Salary, Department_id into sal, deptid from Employees where Employee_Id = 105;
 6     dbms_output.put_line(sal || ' : ' || deptid);
 7     if deptid = 30 then
 8             sal := sal + 3;
 9     elsif deptid = 60 then
10             sal := sal + 6;
11     else
12             sal := sal + 10;
13     end if;
14     dbms_output.put_line(sal || ' : ' || deptid);
15* end;
SQL> /
4800 : 60
4806 : 60

PL/SQL procedure successfully completed.
```

```
 1  declare
 2     deptid number(2);
 3     sal number(10, 2);
 4  begin
 5     select Salary, Department_id into sal, deptid from Employees where Employee_Id = 105;
 6     dbms_output.put_line(sal || ' : ' || deptid);
 7     if deptid = 30 then
 8             sal := sal + 3;
 9     elsif deptid = 60 then
10             sal := sal + 6;
11     else
12             sal := sal + 10;
13     end if;
14     update Employees set Salary = sal where Employee_Id = 105;
15     dbms_output.put_line(sal || ' : ' || deptid);
16* end;
SQL> /
4800 : 60
4806 : 60

PL/SQL procedure successfully completed.
```

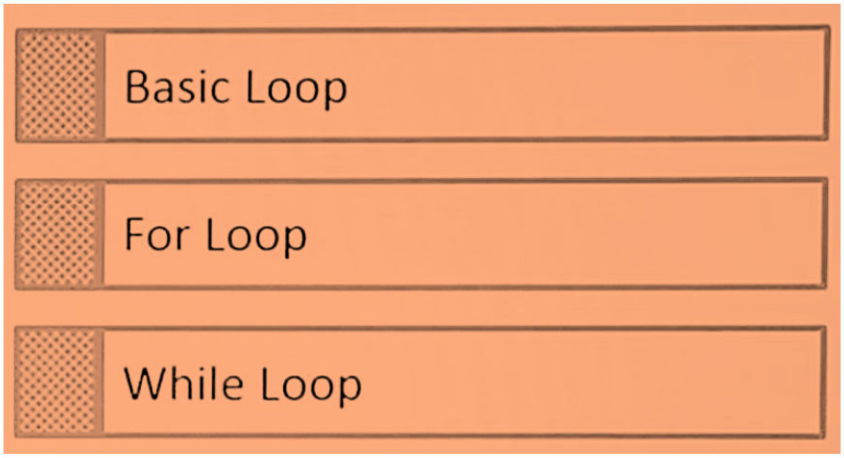
Discussion on the results of the previous Code

```
SQL> /
4800 : 60
4806 : 60

PL/SQL procedure successfully completed.

SQL> /
4806 : 60
4812 : 60

PL/SQL procedure successfully completed.

SQL> /
4812 : 60
4818 : 60

PL/SQL procedure successfully completed.
```

Case statement

```
 1  declare
 2     num number(1) := &Weekday;
 3     dayname varchar2(10);
 4  begin
 5     dayname := Case num
 6               when 1 then 'Monday'
 7               when 2 then 'Tuesday'
 8               when 3 then 'Wednesday'
 9               else 'Sunday'
10     end;
11     dbms_output.put_line(dayname);
12* end;
SQL> /
Enter value for weekday: 2
old   2:         num number(1) := &Weekday;
new   2:         num number(1) := 2;
Tuesday

PL/SQL procedure successfully completed.
```

PL/SQL - Loops



Basic Loop

```
SQL> set serveroutput on
SQL> declare
  2     i number(2);
  3  begin
  4     i := 1;
  5     loop
  6             dbms_output.put_line(i);
  7             i := i + 1;
  8             exit when i > 10;
  9     end loop;
 10  end;
 11  /
1
2
3
4
5
6
7
8
9
10

PL/SQL procedure successfully completed.
```

While Loop

```
SQL> declare
  2     i number(2);
  3  begin
  4     i := 1;
  5     while i <= 10 loop
  6             dbms_output.put_line('i = ' || i);
  7             i := i + 1;
  8     end loop;
  9  end;
 10  /
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10

PL/SQL procedure successfully completed.

SQL>
```

For Loop

```
SQL> begin
  2     for i in 1..10 loop
  3             dbms_output.put_line('i = ' || i);
  4     end loop;
  5  end;
  6  /
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10

PL/SQL procedure successfully completed.

SQL>
```

Print the values in reverse order by putting the reverse keyword

```
SQL> ed
Wrote file afiedt.buf

  1  begin
  2     for i in reverse 1..10 loop
  3              dbms_output.put_line('i = ' || i);
  4     end loop;
  5* end;
SQL> /
i = 10
i = 9
i = 8
i = 7
i = 6
i = 5
i = 4
i = 3
i = 2
i = 1

PL/SQL procedure successfully completed.
```

1.To find sum of two number:

```
declare
a int;
b int;
c int;
begin
a:=&a;
b:=&b;
c:=a+b;
dbms_output.put_line('sum of a and b'||c);
end;
```

2.To find greatest number among two:

```
declare
a int;
b int;
c int;
begin
a:=&a;
b:=&b;
if(a>b)
then
dbms_output.put_line('a is greater');
else
dbms_output.put_line('b is greater');
end if;
end;
```

PL/SQL - Implicit Cursors

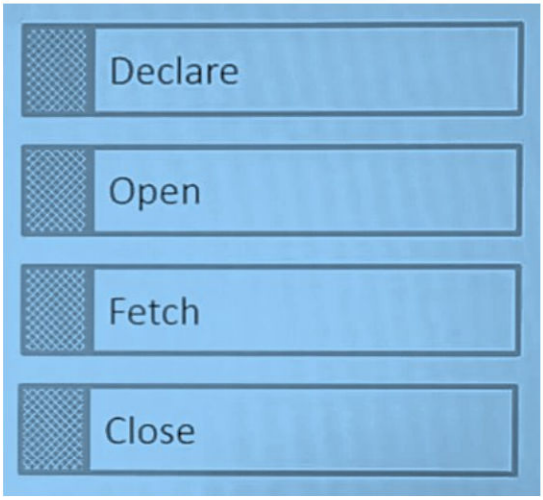
- Is a private SQL work area
- Oracle uses implicit cursors to execute SQL statements and is declared for all DML and PLSQL Select Statement
- Programmers design explicit cursor as per their requirement

```
SQL> declare
  2     cnt number(3);
  3  begin
  4     update Employees set salary = salary + 2 where department_id = 20;
  5     cnt := SQL%RowCount;
  6     dbms_output.put_line(cnt || 'rows updated');
  7  end;
  8  /
2rows updated

PL/SQL procedure successfully completed.

SQL> select count(*) from employees where Department_Id = 20;

  COUNT(*)
----------
         2
```

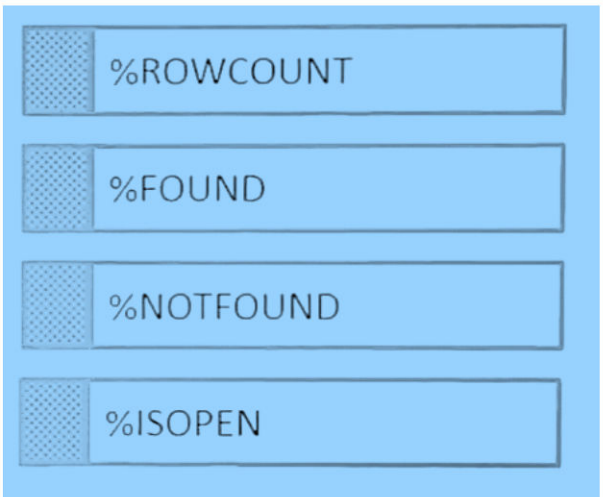
Cursor

A **Cursor** is a temporary memory that is allocated by the database server at the time of performing the **D**ata **M**anipulation **L**anguage operations on a table, such as INSERT, UPDATE and DELETE etc. It is used to retrieve and manipulate data stored in the SQL table. Cursor is used to process multiple records in PL/SQL Program. It is a temporary memory area (context area) where Oracle executes SQL statements. It is also called as Private SQL Area (PSA).

Steps in Explicit Cursors



Cursors Attributes



Cursor Example

```
Declare
name employees. first_name%type;
cursor c1 is
select first_name from employees;
Begin
open c1;
fetch c1 into name;
dbms_output.put_line(name);
close c1;
end;
```

For getting all the first names of employees from the relation.

```
declare
name employees. first_name%type;
cursor c1 is select first_name from employees;
begin
open c1;
Loop
fetch c1 into name;
dbms_output.put_line(name);
exit when (c1%notfound);
end loop;
close c1;
end;
```

Using Basic Loop

```
declare
cursor c1 is select * from employees;
begin
for e in c1 loop
dbms_output.put_line(e.first_name);
end loop;
end;
```

Using For Loop here, open, fetch, and close all the stages are implicitly taken.

```
declare
cursor c1 is select * from employees;
begin
for e in c1 loop
dbms_output.put_line(e.first_name || ' | ' || e.last_name || ' | ' ||  e.salary || ' | ' || e.department_id );
end loop;
end;
/
```

Inline cursor (Cursor FOR Loop with Embedded SELECT)

```
begin
for e in (select * from employees) loop
dbms_output.put_line(e.first_name);
end loop;
end;
/
```

```
SQL> DECLARE
  2     v_empno employees.employee_id%TYPE;
  3     v_ename employees.last_name%TYPE;
  4     CURSOR emp_cursor IS
  5             SELECT employee_id, last_name
  6             FROM employees;
  7  BEGIN
  8     OPEN emp_cursor;
  9     LOOP
 10             FETCH emp_cursor into v_empno, v_ename;
 11             EXIT WHEN emp_cursor%ROWCOUNT > 10 or emp_cursor%NOTFOUND;
 12             DBMS_OUTPUT.PUT_LINE(v_empno || ' : ' || v_ename);
 13     END LOOP;
 14     CLOSE emp_cursor;
 15  END;
 16  /
100 : King
101 : Kochhar
102 : De Haan
103 : Hunold
104 : Ernst
105 : Austin
106 : Pataballa
107 : Lorentz
108 : Greenberg
109 : Faviet

PL/SQL procedure successfully completed.
```

Exception handling in PL/SQL

Syntax

DECLAR

Eexception_name EXCEPTION;

PRAGMA XCEPTION_INIT(exception_name, predefined_error_code);

BEGIN

Executable_statements;

EXCEPTION

WHEN exception_name THEN

exception_handling_code;

END;

```
declare
eid employees.employee_id%type;
sal employees.salary%type;
begin
eid := '&eid';
Select salary into sal from
employees
where employee_id = eid;
dbms_output.put_line(sal);
end;
```

```
Enter value for eid: 101
old   5: eid := '&eid';
new   5: eid := '101';
17000
```

```
SQL> /
Enter value for eid: wqwe
old   5: eid := '&eid';
new   5: eid := 'wqwe';
declare
*
ERROR at line 1:
ORA-06502: PL/SQL: numeric or value error: character to number conversion error
ORA-06512: at line 5
```

When we input the string then it shows the error.

Exception handling …..cont

```
declare
ex exception;
pragma exception_init(ex, -06502);
eid employees.employee_id%type;
sal employees.salary%type;
begin
eid := '&eid';
select salary into sal from employees
where employee_id = eid;
dbms_output.put_line(sal);
exception
when ex then
dbms_output.put_line ('Employee_id takes only numbers as Input. So, provide a number only. Thank You !!!!');
end;
```

```
Enter value for eid: hjdsjhd
old   7: eid := '&eid';
new   7: eid := 'hjdsjhd';
Employee_id takes only numbers as Input. So provide number only. Thank You!!!!

PL/SQL procedure successfully completed.

SQL> /
Enter value for eid: 101
old   7: eid := '&eid';
new   7: eid := '101';
17000

PL/SQL procedure successfully completed.
```

A trigger is a course of action to be taken when any DDL, DML event occurs.

Type of Triggers:

DML Triggers (Table Level)
DDL Triggers (Database Level)

Triggers are created to

Control DML
Implement complex business rule
Implement complex validations
Generate values for primary key
For auditing
Maintain replicas

General Syntax for Triggers:

```
create [or replace ] trigger trigger_name
[before | after | instead of ]
[insert [or] | update [or] | delete] on table_name
[referencing old as 0 new as n]
[for each row] [when (condition)]
[declare <declare section>]
begin
[exception]
end;
```

1. Before Trigger

Before trigger is executed before executing a DML command.

Sequence:
Trigger execution
DML Execution

2. After Trigger

After trigger is executed after executing a DML command.

Sequence:
DML Execution
Trigger execution

DML Triggers

These triggers are executed automatically whenever user performs DML operation on the table.

Levels of DML Triggers

Statement level (default) - executed only once for the statement
Row level - executed once for each row affected by the DML

```
SQL> Set set serveroutput on
SQL> create or replace trigger emptrg2 before update on employees
  2  begin
  3  dbms_output.put_line('Record updated');
  4  end;
  5  /

Trigger created.
```

Example

```
CREATE OR REPLACE TRIGGER xyz
  BEFORE
    INSERT OR
    UPDATE OF salary, department_id OR
    DELETE
  ON XYZ
BEGIN
  CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Inserting');
    WHEN UPDATING('salary') THEN
      DBMS_OUTPUT.PUT_LINE('Updating salary');
    WHEN UPDATING('department_id') THEN
      DBMS_OUTPUT.PUT_LINE('Updating department ID');
    WHEN DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Deleting');
  END CASE;
END;
/
```

Fire the trigger for each row

```
SQL> create or replace trigger xyztrg
before update on xyz for each row
begin
dbms_output.put_line('Record updated');
end;
/

SQL> update xyz set salary = salary * 1.2 where salary < 10000;
Record updated
Record updated
Record updated
Record updated
Record updated

4 rows updated.
```

DDL Trigger

```
SQL>create or replace trigger hrtr1
before drop on hr. schema
begin
dbms_output.put_line('Your table has been
Dropped');
end;

Trigger created.
```

```
SQL> drop table ab;
Your table has been Dropped

Table dropped.
```

Bind Variables In PL/SQL

```
SQL> select * from accounts;

      ACNO NAME           BALANCE
---------- ---------- ----------
       100 ravi            25000
       101 Rohit           55000
       102 Ram             65000
       103 Mohit           95000
       104 shobhit         55000
       105 Geeta           60000

6 rows selected.
```

```
SQL> desc trans
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------
----------
 TID                                                NUMBER(5)
 ACNO                                               NUMBER(6)
 TDATE                                              DATE
 AMOUNT                                             NUMBER(8)
 BALANCE                                            NUMBER(7)

SQL>
```

Create a sequence and then use it in the trigger

```
SQL> create sequence ac_seq;
```

```
create or replace trigger actr2
before update on accounts for each row
begin
insert into trans
(tid, acno, tdate, amount, balance) values
(ac_seq.nextval, :new.acno, sysdate,
abs(:new.balance - :old.balance), :new.balance);
end; /
```

Now we will update the record in the accounts table and see the working of the trigger actr2 and show the record of the trans table.

```
SQL> update accounts set balance = balance + 10000
  2  where acno = 104;

1 row updated.

SQL> select * from trans;

       TID       ACNO TDATE         AMOUNT    BALANCE
---------- ---------- --------- ---------- ----------
         1        104 16-MAY-25      10000      55000
```