|  |  |
| --- | --- |
|  |  |

DECLARE

var number(3) := 50;

BEGIN

**IF** (var = 10) THEN

dbms\_output.put\_line('Value of var is 10');

ELSIF (var = 20) THEN

dbms\_output.put\_line('Value of var is 20');

ELSIF (var = 30) THEN

dbms\_output.put\_line('Value of var is 30');

**ELSE**

dbms\_output.put\_line('None of the above condition is true.');

END **IF**;

dbms\_output.put\_line('Exact value of var is: '|| var);

END;

/

DECLARE

nameChar **char**(1) := 'J';

BEGIN

**CASE** nameChar

when 'B' then dbms\_output.put\_line('B');

when 'R' then dbms\_output.put\_line('R');

when 'S' then dbms\_output.put\_line('S');

when 'V' then dbms\_output.put\_line('V');

when 'J' then dbms\_output.put\_line('J');

**else** dbms\_output.put\_line('No such name');

END **CASE**;

END;

/

DECLARE

num NUMBER := 1;

BEGIN

LOOP

DBMS\_OUTPUT.PUT\_LINE(num);

**IF** num = 10 THEN

EXIT;

END **IF**;

num := num+1;

END LOOP;

END;

/

DECLARE

num NUMBER := 1;

BEGIN

**WHILE** num <= 10

LOOP

DBMS\_OUTPUT.PUT\_LINE(num);

num := num+1;

END LOOP;

END;

/

DECLARE

BEGIN

**FOR** var IN 1..10

LOOP

DBMS\_OUTPUT.PUT\_LINE(var);

END LOOP;

END;

/

DECLARE

num NUMBER := 0;

BEGIN

**WHILE** num < 10

LOOP

num := num +1;

**IF** num = 5 THEN

**CONTINUE**;

END **IF**;

DBMS\_OUTPUT.PUT\_LINE(num);

END LOOP;

END;

/

DECLARE

i number(1);

j number(1);

BEGIN

<< outer\_loop >>

**FOR** i IN 1..5 LOOP

<< inner\_loop >>

**FOR** j IN 1..5 LOOP

dbms\_output.put\_line('i is: '|| i || ' and j is: ' || j);

END loop inner\_loop;

END loop outer\_loop;

END;

/

DECLARE

num number := 1;

BEGIN

<<loop1>>

-- **while** loop execution

**WHILE** num <= 10 LOOP

dbms\_output.put\_line ('Value of num: ' || num);

num := num + 1;

**IF** num = 5 THEN

num := num + 1;

**GOTO** loop1;

END **IF**;

END LOOP;

END;

/

Stored Procedure:

CREATE OR REPLACE PROCEDURE hello\_world

AS

BEGIN

dbms\_output.put\_line('Hello World!');

END;

/

Before below, you should already have a table called students

CREATE OR REPLACE PROCEDURE add\_student(rollNo IN NUMBER, name IN VARCHAR2)

IS

BEGIN

insert into students values(rollNo,name);

END;

/

Exec/execute add\_student;

At times the below also works:

Begin

Add\_student;

End;

/

Drop procedure add\_student;

Function:

create or replace function getMultiple(num1 in number, num2 in number)

**return** number

is

num3 number(8);

begin

num3 :=num1\*num2;

**return** num3;

end;

/

Implicit cursor: ROWCOUNT is an implicit cursor.

You should have an employees table in prior!

DECLARE var\_rows number(2);

BEGIN

UPDATE employees

SET salary = salary + 2000;

**IF** SQL%NOTFOUND THEN

dbms\_output.put\_line('No record updated.');

ELSIF SQL%FOUND THEN

var\_rows := SQL%ROWCOUNT;

dbms\_output.put\_line(var\_rows || ' records are updated.');

END **IF**;

END;

/

Explicit cursor:

Should have a students table already!

CURSOR cur\_students IS

SELECT rollNo, name, address FROM students;

Using explicit cursor in SP (Stored procedure):

DECLARE

s\_rollNo students.rollNo%type;

s\_name students.name%type;

s\_address students.address%type;

CURSOR cur\_students is

SELECT rollNo, name, address FROM students;

BEGIN

OPEN cur\_students;

LOOP

FETCH cur\_students into s\_rollNo, s\_name, s\_address;

EXIT WHEN cur\_students%notfound;

dbms\_output.put\_line(s\_rollNo || ' ' || s\_name || ' ' || s\_address);

END LOOP;

CLOSE cur\_students;

END;

/

Exceptions: System defined and user defined exceptions:

DECLARE

s\_rollNo students.rollNo%type := 10;

s\_name students.name%type;

s\_address students.address%type;

BEGIN

SELECT rollNo, name, address FROM students WHERE rollNo = s\_rollNo;

dbms\_output.put\_line(s\_rollNo || ' ' || s\_name || ' ' || s\_address);

EXCEPTION

WHEN no\_data\_found THEN

dbms\_output.put\_line('No such student!');

WHEN others THEN

dbms\_output.put\_line('Error!');

END;

/

Trigger: to automatically trigger some stored procedure!

Select \* from employees;

|  |
| --- |
| EMP\_ID NAME AGE ADDRESS SALAR |

CREATE OR REPLACE TRIGGER show\_salary\_difference

BEFORE DELETE OR INSERT OR UPDATE ON employees

**FOR** EACH ROW

WHEN (**NEW**.EMP\_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;

/

*CREATE OR REPLACE PROCEDURE employer\_details*

*2> IS*

*3> CURSOR emp\_cur IS*

*4> SELECT first\_name, last\_name, salary FROM emp\_tbl;*

*5> emp\_rec emp\_cur%rowtype;*

*6> BEGIN*

*7> FOR emp\_rec in sales\_cur*

*8> LOOP*

*9> dbms\_output.put\_line(emp\_cur.first\_name || ' ' ||emp\_cur.last\_name*

*10> || ' ' ||emp\_cur.salary);*

*11> END LOOP;*

*12>END;*

*13> /*

Create emp\_tbl first:

*1> CREATE OR REPLACE PROCEDURE emp\_salary\_increase*

*2> (emp\_id IN emptbl.empID%type, salary\_inc IN OUT emptbl.salary%type)*

*3> IS*

*4> tmp\_sal number;*

*5> BEGIN*

*6> SELECT salary*

*7> INTO tmp\_sal*

*8> FROM emp\_tbl*

*9> WHERE empID = emp\_id;*

*10> IF tmp\_sal between 10000 and 20000 THEN*

*11> salary\_inout := tmp\_sal \* 1.2;*

*12> ELSIF tmp\_sal between 20000 and 30000 THEN*

*13> salary\_inout := tmp\_sal \* 1.3;*

*14> ELSIF tmp\_sal > 30000 THEN*

*15> salary\_inout := tmp\_sal \* 1.4;*

*16> END IF;*

*17> END;*

*18> /*

Execute the above procedure using below:

*1> DECLARE*

*2> CURSOR updated\_sal is*

*3> SELECT empID,salary*

*4> FROM emp\_tbl;*

*5> pre\_sal number;*

*6> BEGIN*

*7> FOR emp\_rec IN updated\_sal LOOP*

*8> pre\_sal := emp\_rec.salary;*

*9> emp\_salary\_increase(emp\_rec.empID, emp\_rec.salary);*

*10> dbms\_output.put\_line('The salary of ' || emp\_rec.empID ||*

*11> ' increased from '|| pre\_sal || ' to '||emp\_rec.salary);*

*12> END LOOP;*

*13> END;*

*14> /*

*What is a Function in PL/SQL?*

*A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.*

*CREATE OR REPLACE FUNCTION employer\_details\_func*

*2> RETURN VARCHAR(20);*

*3> IS*

*5> emp\_name VARCHAR(20);*

*6> BEGIN*

*7> SELECT first\_name INTO emp\_name*

*8> FROM emp\_tbl WHERE empID = '100';*

*9> RETURN emp\_name;*

*10> END;*

*11> /*

*Calling:*

*dbms\_output.put\_line(employer\_details\_func);*

*Good example on Triggers:*

[*PL/SQL Tutorial- PL/SQL Triggers (plsql-tutorial.com)*](http://plsql-tutorial.com/plsql-triggers.htm)

PLSQL Cursor example:

[*PL/SQL Cursor By Practical Examples (oracletutorial.com)*](https://www.oracletutorial.com/plsql-tutorial/plsql-cursor/)

DECLARE

CURSOR kpi\_det IS SELECT emp\_name FROM emp;

lv\_emp\_name emp.emp\_name%type;

BEGIN

OPEN kpi\_det;

LOOP

FETCH kpi\_det INTO lv\_emp\_name;

IF kpi\_det%NOTFOUND

THEN

EXIT;

END IF;

Dbms\_output.put\_line(‘Employee Fetched:‘||lv\_emp\_name);

END LOOP;

Dbms\_output.put\_line(‘Total rows fetched is‘||kpi\_det%R0WCOUNT);

CLOSE kpi\_det;

END:

/

Exceptions:

DECLARE

s\_rollNo students.rollNo%type := 10;

s\_name students.name%type;

s\_address students.address%type;

BEGIN

SELECT rollNo, name, address FROM students WHERE rollNo = s\_rollNo;

dbms\_output.put\_line(s\_rollNo || ' ' || s\_name || ' ' || s\_address);

EXCEPTION

WHEN no\_data\_found THEN

dbms\_output.put\_line('No such student!');

WHEN others THEN

dbms\_output.put\_line('Error!');

END;

/

DECLARE

s\_rollNo students.rollNo%type := &ss\_rollNo;

s\_name students.name%type;

s\_address students.address%type;

-- user defined exception

ex\_invalid\_rollNo EXCEPTION;

BEGIN

IF c\_id <= 0 THEN

RAISE ex\_invalid\_rollNo;

ELSE

SELECT rollNo, name, address FROM students WHERE rollNo = s\_rollNo;

dbms\_output.put\_line(s\_rollNo || ' ' || s\_name || ' ' || s\_address);

END IF;

EXCEPTION

WHEN ex\_invalid\_rollNo THEN

dbms\_output.put\_line('rollNo must be greater than zero!');

WHEN no\_data\_found THEN

dbms\_output.put\_line('No such student!');

WHEN others THEN

dbms\_output.put\_line('Error!');

END;

/

## PL/SQL exception categories

PL/SQL has three exception categories:

* Internally defined exceptions are errors which arise from the Oracle Database environment. The runtime system raises the internally defined exceptions automatically. ORA-27102 (out of memory) is one example of Internally defined exceptions. Note that Internally defined exceptions do not have names, but an error code.
* Predefined exceptions are errors which occur during the execution of the program. The predefined exceptions are internally defined exceptions that PL/SQL has given names e.g., NO\_DATA\_FOUND, TOO\_MANY\_ROWS.
* User-defined exceptions are custom exception defined by users like you. User-defined exceptions must be raised explicitly.

Built-in packages:

|  |
| --- |
| DBMS\_OUTPUT |
| UTL\_FILE |
| DBMS\_LOB |

SQL> -- Simple PLSQL to open a file,

SQL> -- write two lines into the file,

SQL> -- and close the file

SQL> declare

  2    fhandle  utl\_file.file\_type;

  3  begin

  4    fhandle := utl\_file.fopen(

  5                  'UTL\_DIR'     -- File location

  6                , 'test\_file.txt' -- File name

  7                , 'w' -- Open mode: w = write.

  8                    );

  9

 10    utl\_file.put(fhandle, 'Hello world!'

 11                        || CHR(10));

 12    utl\_file.put(fhandle, 'Hello again!');

 13

 14    utl\_file.fclose(fhandle);

 15  exception

 16    when others then

 17      dbms\_output.put\_line('ERROR: ' || SQLCODE

 18                        || ' - ' || SQLERRM);

 19      raise;

 20  end;

 21  /

Additional details:

[Oracle UTL\_FILE Package with Example - Know Program](https://www.knowprogram.com/oracle/oracle-utl_file/)

DBMS\_LOB is a package that provides the suborograms to the user in order to operate on BLOBs, CLOBs and NCLOBs. This package can also be used to manipulate and access the LOB values in both external and internal locations of storage. This package provides the read-only operations for BFILEs.

declare  
a\_new1 clob;  
a\_new2 clob;  
begin  
a\_new1 := 'raghav';  
select pdetails into a\_new2 from products where pid = 1001;  
dbms\_output.put\_line('before append, value of a\_new1 is ' || a\_new1);  
dbms\_output.put\_line('before append value of a\_new2 is ' || a\_new2);  
dbms\_lob.append(a\_new1, a\_new2);  
dbms\_output.put\_line('After append value of a\_new1 is ' || a\_new1);  
dbms\_output.put\_line('After append value of a\_new2 is ' || a\_new2);  
insert into products values (1002, a\_new1);  
end;