- %TYPE

%TYPE is used to declare variables with relation to the data type of a column in an existing table:

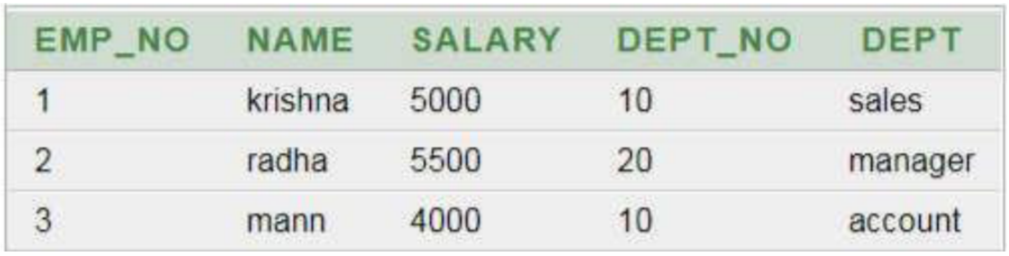
Write a program to use of %type attribute.

**Create table....**

```
create table emp1
(emp_no number(5),
name varchar2(10),
salary number(5),
dept_no number(5),
dept varchar2(20));
```

**Now Insert records into the table ....**

```
insert into emp1 values(1,'krishna',5000,10,'sales');
```

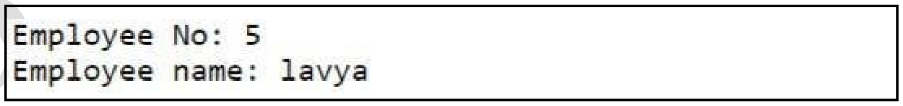


| EMP_NO | NAME | SALARY | DEPT_NO | DEPT |
|---|---|---|---|---|
| 1 | krishna | 5000 | 10 | sales |
| 2 | radha | 5500 | 20 | manager |
| 3 | mann | 4000 | 10 | account |

**PL/SQL block**

```
Declare
  eno emp1.emp_no %TYPE;
  ename emp1.name %TYPE;
Begin
  eno:= '5';
  ename:= 'lavya';
  dbms_output.put_line('Employee No: ' || eno);
  dbms_output.put_line('Employee name: ' || ename);
End;
```

**Output:**



```
Employee No: 5
Employee name: lavya
```

## %ROWTYPE

The %ROWTYPE attribute, used to declare PL/SQL variables of type record with fields that correspond to the columns of a table or view, is supported by the data server.

**Create table....**

```
select *From emp
```

**PL/SQL block**

```
declare
        myr emp %rowtype;
begin
        select id,name,salary into myr.id,myr.name,myr.salary from emp
WHERE ROWNUM = 1;
        dbms_output.put_line('id is:'||myr.id);
        dbms_output.put_line('name is:'||myr.name);
        dbms_output.put_line('salary is:'||myr.salary);
end;
```

**Output :**

```
id is:12
name is:ABC
salary is:1000

Statement processed.
```

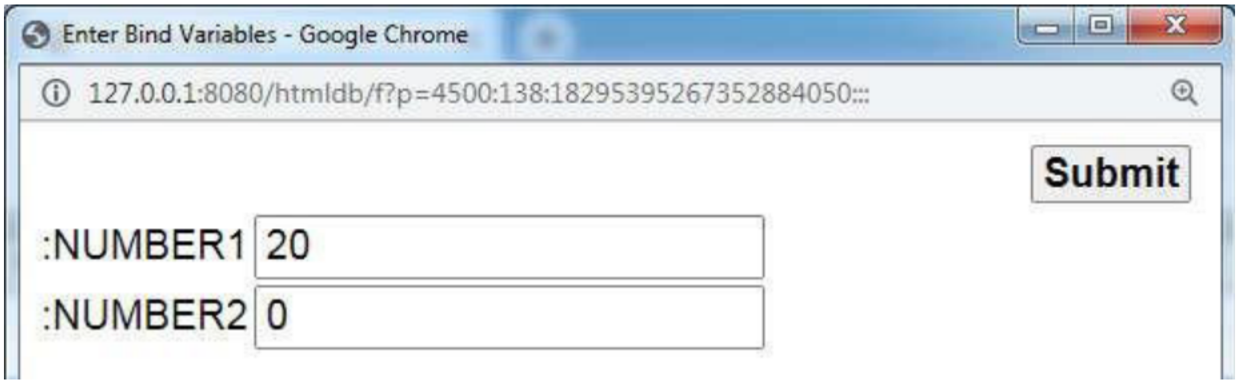
## Exception Handling

-  An exception is an error condition during a program execution.

**PL/SQL block**

```
Declare
    no1 number(10);
    no2 number(10);
    ans number(10);
Begin
    no1:=:number1;
    no2:=:number2;
    ans:= no1/no2;
    dbms_output.put_line ('Division is = '|| ans);
Exception
    When zero_divide then
        dbms_output.put_line ('you have entered no2 as zero');
        dbms_output.put_line ('Please entered another value');
End;
```

**Output :**


Enter Bind Variables - Google Chrome

127.0.0.1:8080/htmldb/f?p=4500:138:1829539526735288405O:::

:NUMBER1 20

:NUMBER2 0

```
You have entered no2 as zero
Please enter another value
```

'PL/SQL - Packages'.

- PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms –

- *Functions* – These subprograms return a single value; mainly used to compute and return a value.
- *Procedures* – These subprograms do not return a value directly; mainly used to perform an action.

## PL/SQL block

## Creating a Procedure

### Program-1

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
  dbms_output.put_line('Hello World!');
END;
/
```

### Executing a called from another PL/SQL block

```
BEGIN
  greetings;
END;
/
```

### Output :

Hello World

PL/SQL procedure successfully completed.

**Program-2 [Procedure (with in parameter) ]**

```
Create or replace procedure p1(a in number)is
   ans number;
Begin
   ans:= a*2;
   dbms_output.put_line ('The Answer is:'|| ans);
End;
```

**Executing a called from another PL/SQL block**

```
Begin
  p1(5);
End;
```

**Output :**

```
                    The Answer is:10

                    Statement processed.
```

**Deleting a Procedure**

**DROP PROCEDURE procedure-name;**

## Creating a Function

## create table customers

create table customers (customers_name varchar2(10))

insert into customers values ('Raj');

insert into customers values ('Rajesh');

insert into customers values ('Rajendra');

```
CREATE OR REPLACE FUNCTION totalCustomers
  RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total FROM customers;
  RETURN total;
END;
/
```

Calling a Function

```
DECLARE
  c number(2);
BEGIN
  c := totalCustomers();
  dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result –

```
Total no. of Customers: 6

PL/SQL procedure successfully completed.
```

## --Package

The PL/SQL package is the group of the related procedures, functions, variables, and other bundled constructs for providing the **modularity** and **organized approach** to application development.

**-- Create a package to display appropriate message.**

-- Package Definition:

```
Create or replace package pack1 as
  Procedure p1;
  Function f1 return varchar;
End;
```

```
Package created.
```

–Package body

```
Create or replace package body pack1 as
     Procedure p1 is
Begin
     dbms_output.put_line ('Hi this is from procedure');
End p1;

Function f1 return varchar is Message varchar(50);
Begin
     message:='Hello this is a function';
     Return (message);
End f1;
End pack1;
```

```
Package Body created.
```

```
–PL/SQL block
Declare
  --msg varchar(50);
Begin
  --msg:=pack1.f1();
    dbms_output.put_line(pack1.f1 ());
    pack1.p1 ();
End;
```

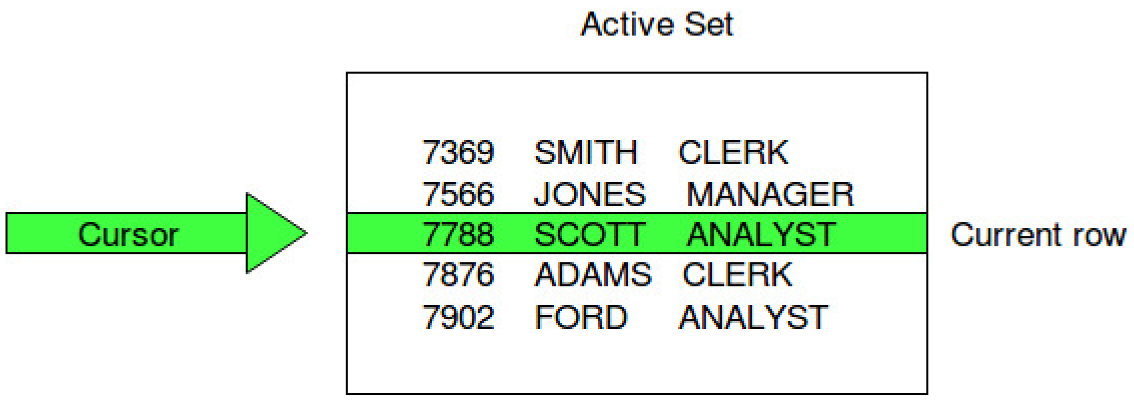
```
Hello this is a function
Hi this is from procedure

Statement processed.
```

## PL/SQL Cursors

The cursor is used to retrieve data one row at a time from the results set, unlike other SQL commands that operate on all rows at once.
Cursors update table records in a singleton or row-by-row manner.

The Data that is stored in the Cursor is called the **Active Data Set**. Oracle DBMS has another predefined area in the main memory Set, within which the cursors are opened. Hence the size of the cursor is limited by the size of this pre-defined area.

## Cursor Functions



```
7369   SMITH   CLERK
7566   JONES   MANAGER
7788   SCOTT   ANALYST
7876   ADAMS   CLERK
7902   FORD    ANALYST
```

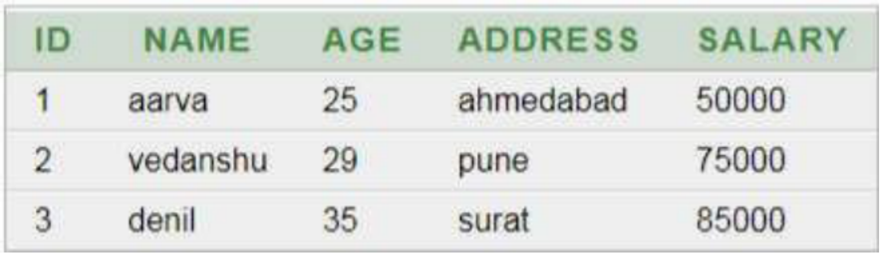
## -- Trigger

A PL/SQL trigger is a named database object that encapsulates and defines a set of actions that are to be performed in response to an insert, update, or delete operation against a table. Triggers are created using the PL/SQL CREATE TRIGGER statement. Types of triggers (PL/SQL)

**--Create a trigger to display salary changes in the customer table.**

--First create customer table and insert records.

```
create table customers
(id number(3),
name varchar2(20),
age number(2),
address varchar2(20),
salary number(10));
```



| ID | NAME | AGE | ADDRESS | SALARY |
|---|---|---|---|---|
| 1 | aarva | 25 | ahmedabad | 50000 |
| 2 | vedanshu | 29 | pune | 75000 |
| 3 | denil | 35 | surat | 85000 |

```
insert into customers values(1,'aarva',25,'ahmedabad',50000);
insert into customers values(1,aaa,26,'surat',60000);

--Create a trigger
CREATE OR REPLACE TRIGGER display_salary_changes
   BEFORE DELETE OR INSERT OR UPDATE ON customers
   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;
```

```
Trigger created.
```

```
--PL/SQL block to update customer table..
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers SET salary = salary + 5000;
   IF sql%notfound THEN
      dbms_output.put_line('no customers updated');
   ELSIF sql%found THEN
       total_rows := sql%rowcount;
   dbms_output.put_line( total_rows || ' customers updated ');
   END IF;
END;
```

```
Old salary: 50000
New salary: 55000
Salary difference: 5000
Old salary: 60000
New salary: 65000
Salary difference: 5000
2 customers updated
```

## PLSQL TABLES [Nested table]

Create a nested table called dept and use it in to query.

```
Create or replace type dept1 as table of varchar2 (10);

-- Now create a table
Create table emp1_1 (emp_no number (10), ename varchar2 (10),
deptartment dept1)
Nested table deptartment store as dept_tab;

-- After it insert a record in it
Insert into emp1_1 values (1,'aaa', dept1 ('Account','Clerk'));
-- After it show the record
```

**Output:**

```
Select * from emp1_1;

EMP_NO        ENAME              DEPARTMENT
===========================================
    1            AAA         DEPT1 ('Account', 'Clerk')
```

## PLSQL TABLES [Varray]

**create a varray called marks_va of size 5 & apply on the table**

```
Create or replace type marks_va as varray (5) of number (5);

-- After varray created then create a table student
Create table student (std_no number (10) primary key, name varchar2 (15),
marks
marks_va);

-- After table is created then insert a row in the table
Insert into student values (1,'aaa', marks_va (50, 60, 70));
insert into student values(2,'bbb',marks_va(50,60,65,70,75));

-- After row is inserted then
Select * from student;
```

**Output:**

```
STD_NO   NAME    MARKS
==============================
1           AAA    MARKSVA (50, 60, 70)
```

**Varray Vs. Nested Tables:**

<u>Similarities</u>

- Both types allow access to individual elements using subscript notation.
- Both types can be stored in database tables.

<u>Difference</u>

- Varray have a maximum size, while nested tables do not.
- Varray are stored inline with the containing table while nested tables are stored in a separate table, which can have different storage characteristics.
- When sorted in the database varray retain the ordering and subscript values for the elements, while nested tables do not.
- Individual elements can be deleted from a nested table, which cause the size of the table to shrink. A varray is always a constant size however.