DBMS LAB Assignment-4

ERD / Entity Relationship Model

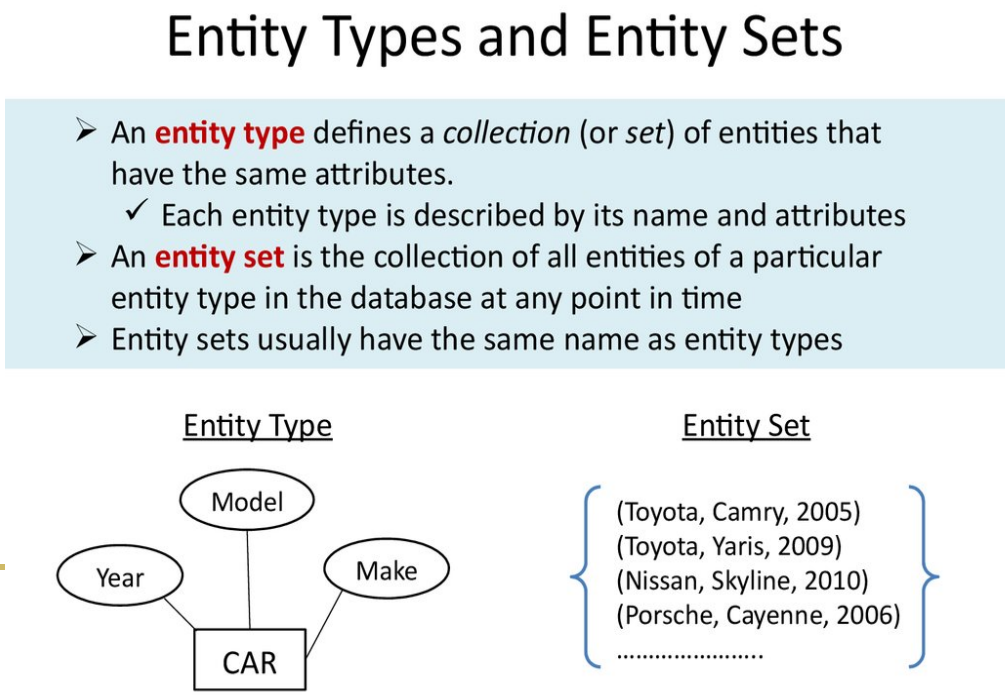
- Data Model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities.
- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

Components

- An entity is a real-world item or concept or event - that have data stored about it, exists on its own and can be distinctly identified .
- An **Entity** is an object of Entity Type .

An entity might be:

- An object with physical existence. E.g. a lecturer, a student, a car
- An object with conceptual existence. E.g. a course, a job, a position



Entity Types and Entity Sets

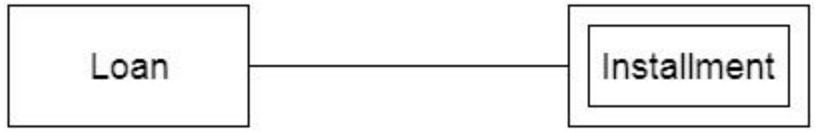
- ➢ An **entity type** defines a *collection* (or *set*) of entities that have the same attributes.
  - ✓ Each entity type is described by its name and attributes
- ➢ An **entity set** is the collection of all entities of a particular entity type in the database at any point in time
- ➢ Entity sets usually have the same name as entity types

Entity Type

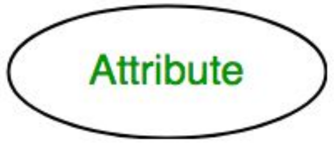
Entity Set

{ (Toyota, Camry, 2005)
(Toyota, Yaris, 2009)
(Nissan, Skyline, 2010)
(Porsche, Cayenne, 2006)
…………………… }

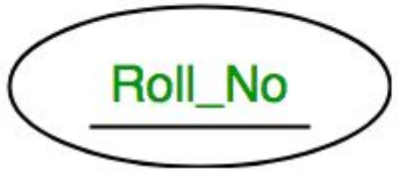
Weak Entity



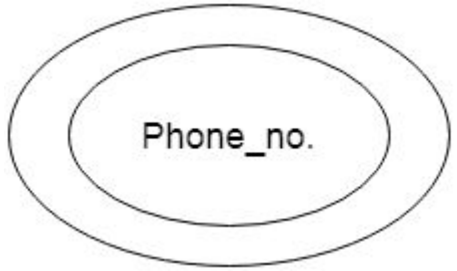
Attribute



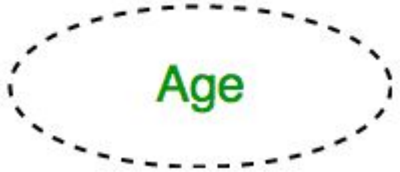
Key Attribute



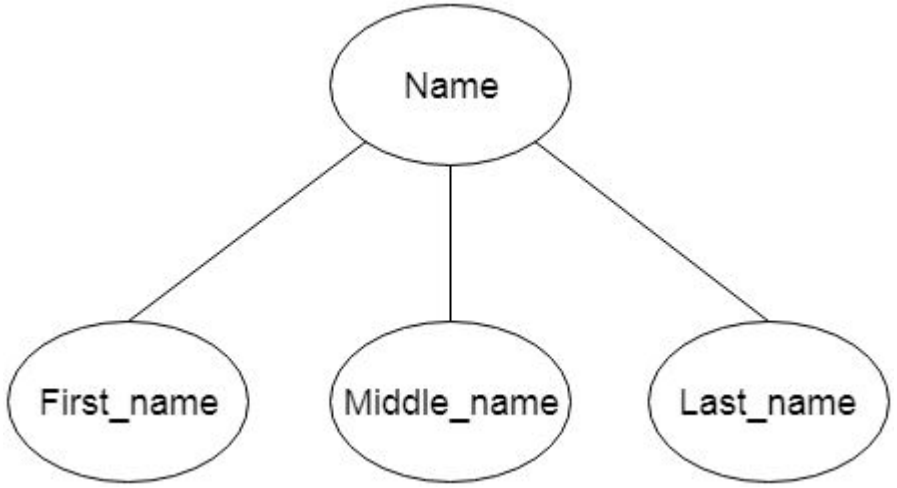
Multivalued Attribute



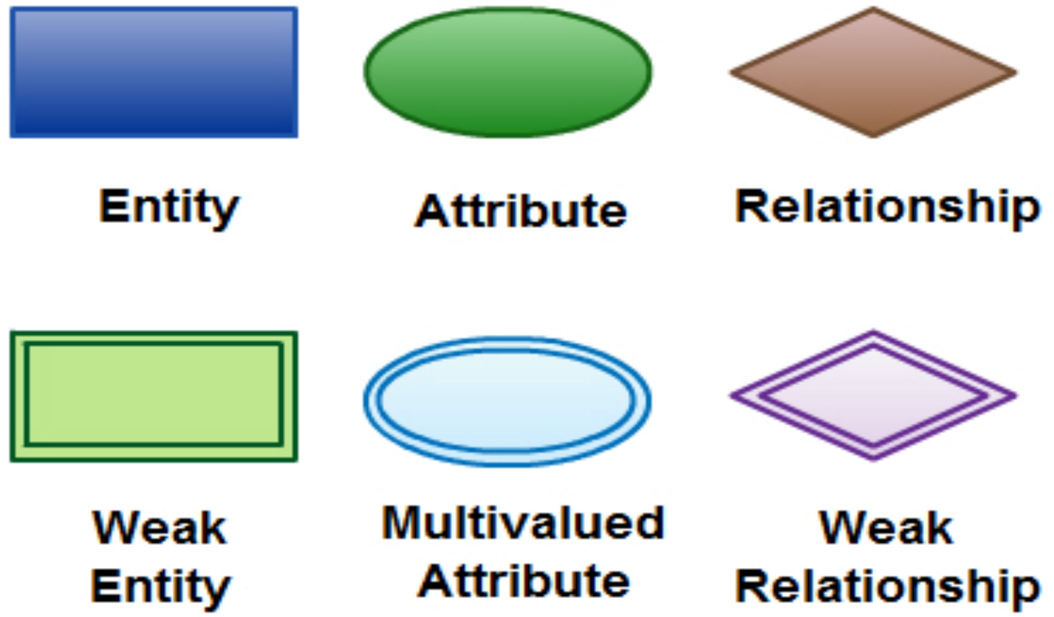
Derived Attribute



Composite Attribute

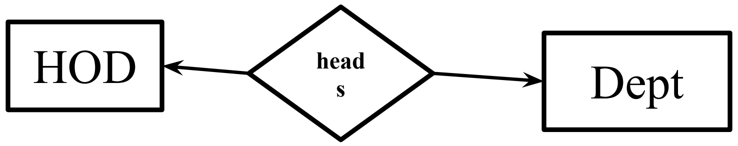


Weak entity type doesn't have a key attribute. Weak entity type can't be identified on its own. It depends upon some other strong entity for its distinct identity.

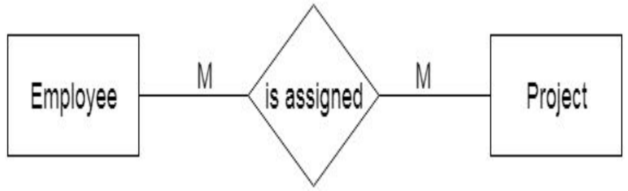

Entity
Attribute
Relationship
Weak Entity
Multivalued Attribute
Weak Relationship

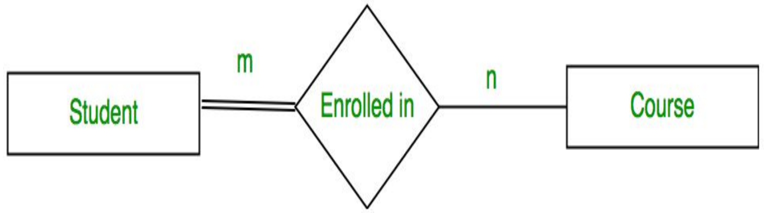
Relationship: A relationship type represents the **association between entity types.**
Cardinality: the possible number of occurrences in one entity which is associated with the number of occurrences in another.
A set of relationships of same type is known as relationship set.
Participation Constraint: (Minimum Cardinality)
Participation Constraint is applied on the entity participating in the relationship set.
Total Participation – Each entity in the entity set **must participate** in the relationship.
Partial Participation – The entity in the entity set **may or may NOT participat**e in the relationship.

One-to-One Relationship



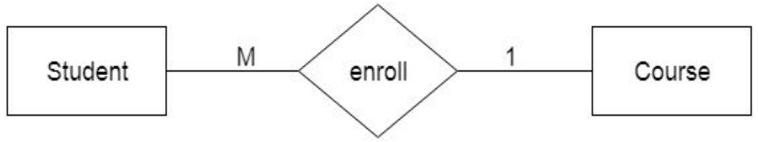
Many-to-many relationship



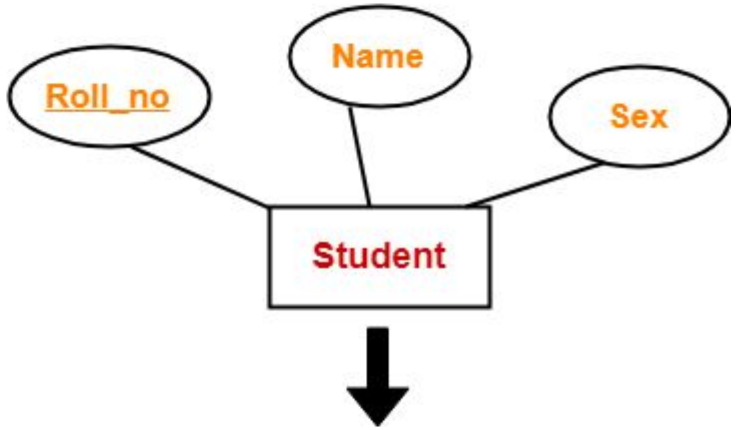


- If each student must enroll in a course, the participation of student will be total.
- If some courses are not enrolled by any of the student, the participation of course will be partial

Many-to-one relationship

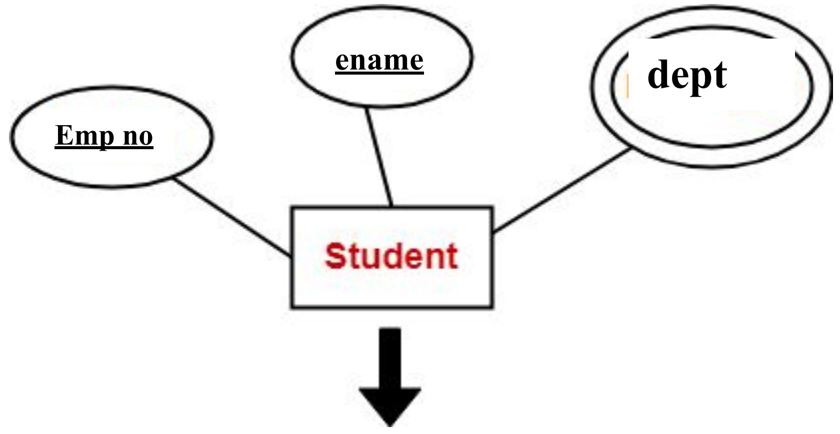


Convert ERD to Relational Model



Roll_no	Name	Sex

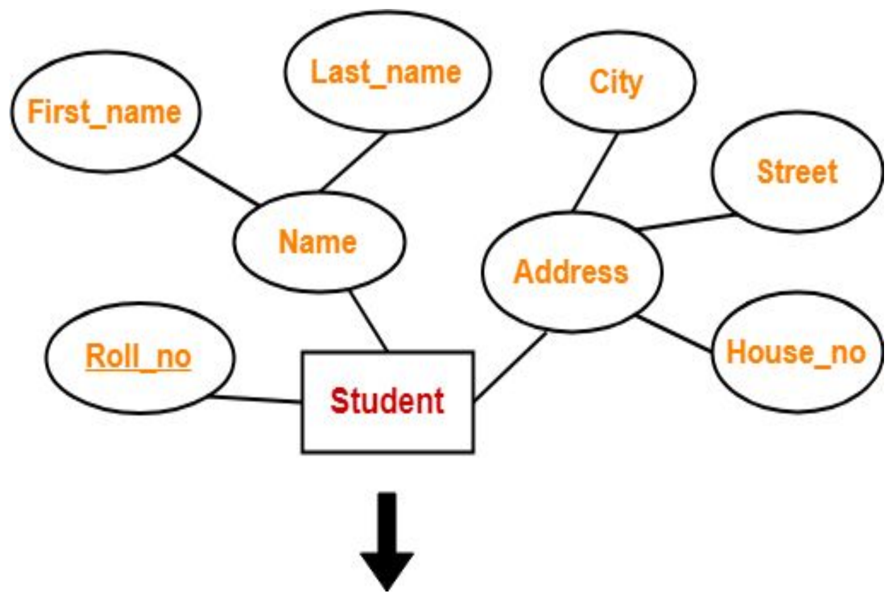
Schema : Student (Roll_no , Name , Sex)



Emp no	ename
1	A
2	B
3	C

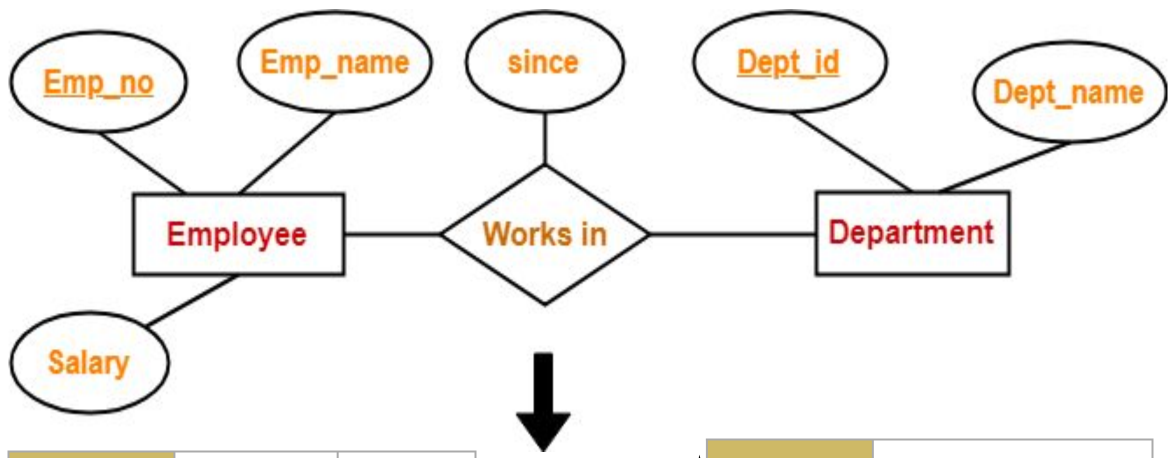
Emp no	dept
1	CS
1	BSC
2	MA

Translating Relationship Set into a Table



<u>**Roll_no**</u>	**First_name**	**Last_name**	**House_no**	**Street**	**City**

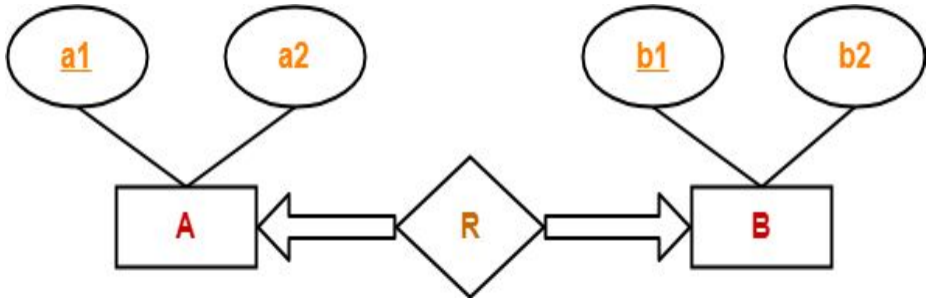
Schema : Student (<u>Roll_no</u> , First_name , Last_name , House_no , Street , City)



<u>**Emp_no**</u>	**Emp_name**	**salary**

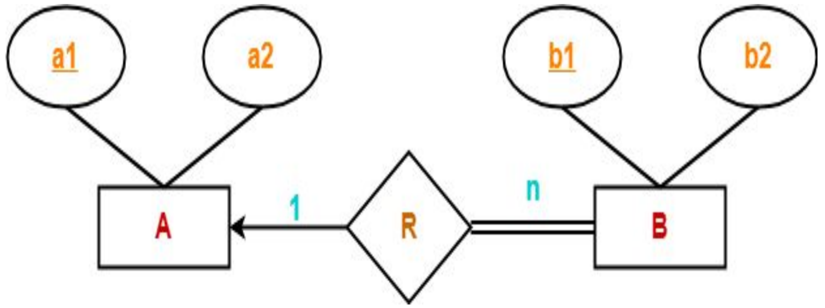
<u>**Dept_id**</u>	**Dept_name**

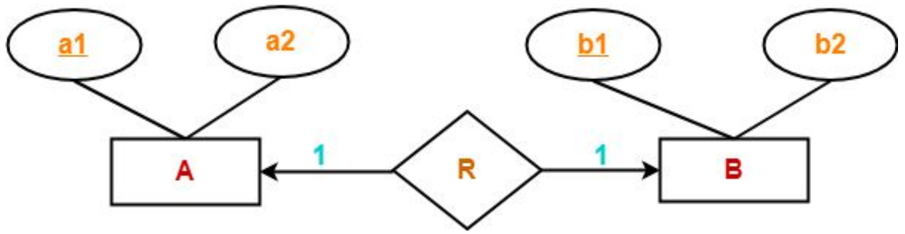
<u>**Emp_no**</u>	<u>**Dept_id**</u>	**since**



ARB (<u>a1</u> <u>b1</u>, a2, b2)

If there is a key constraint from both the sides of an entity set with total participation, then that binary relationship is represented using only single table.





Way-01:
AR (<u>a1</u> , a2 , b1)
B (<u>b1</u> , b2)

Way-02:
A (<u>a1</u> , a2)
BR (<u>b1</u> , b2, a1)

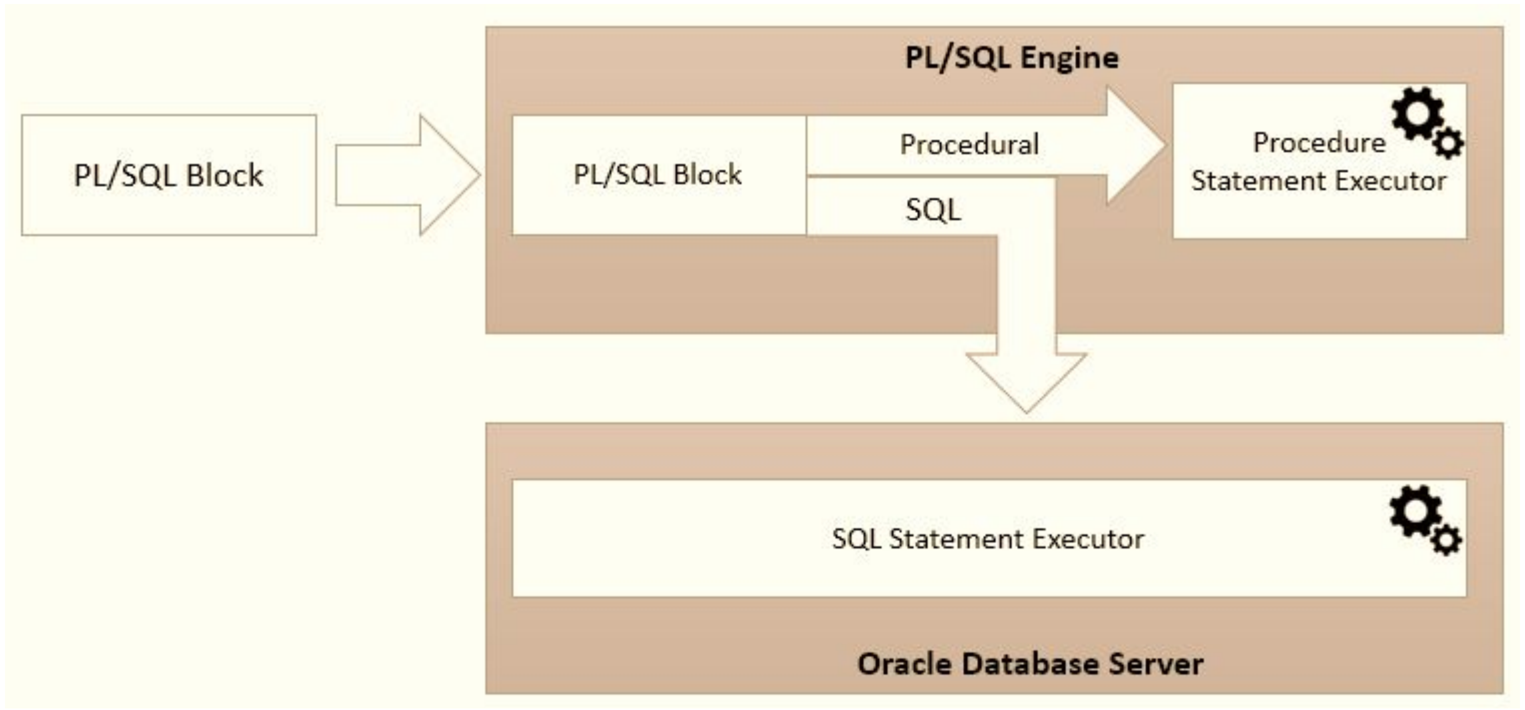
A (<u>a1</u>,a2)
BR (<u>b1</u> , b2, a1)

Because of total participation, foreign key a1 has acquired NOT NULL constraint, so it can’t be null now.

PL/SQL

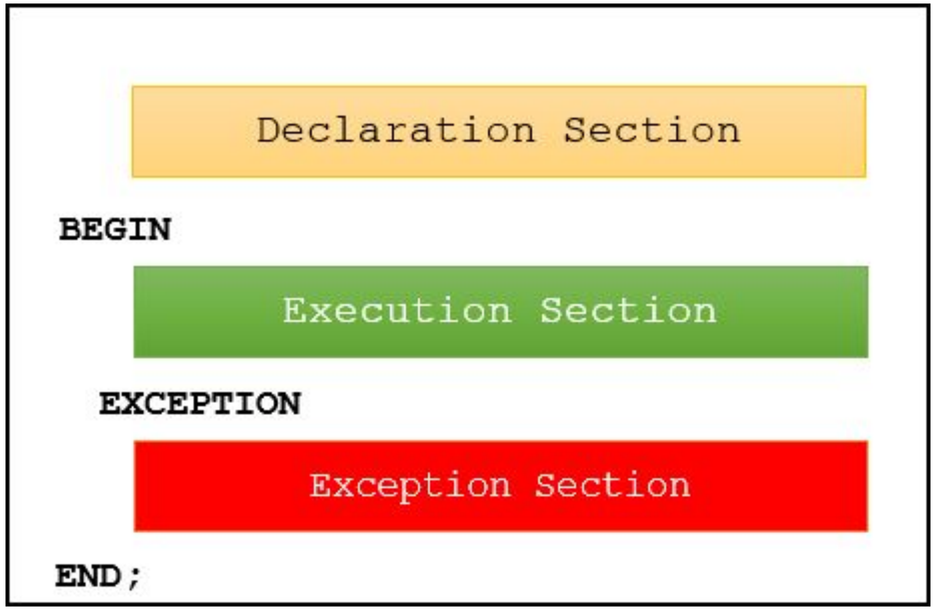
- PL/SQL stands for “Procedural Language extensions to the Structured Query Language”
- PL/SQL adds many procedural constructs to SQL language to overcome some limitations of SQL.
- Besides, PL/SQL provides a more comprehensive programming language solution for building mission-critical applications on Oracle Databases.

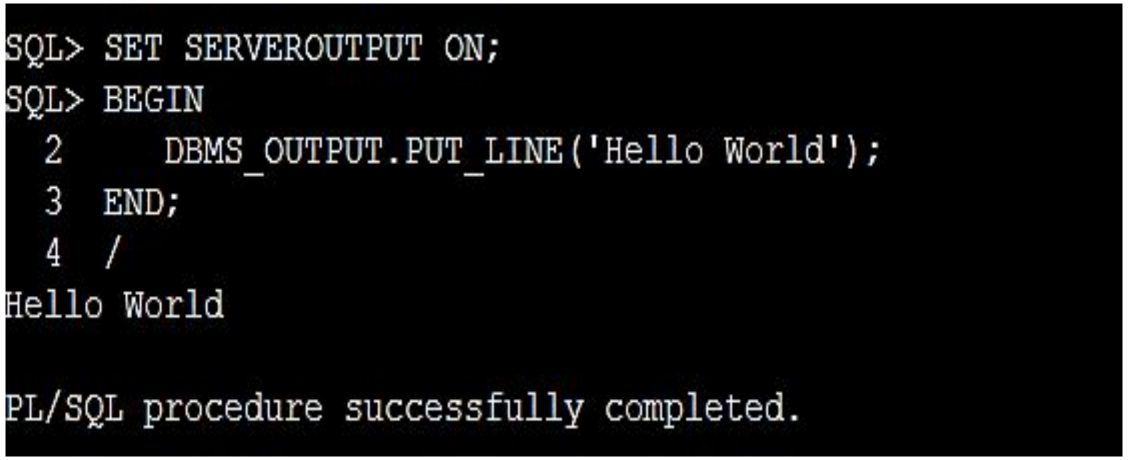
Architecture



https://www.oracletutorial.com/plsql-tutorial/what-is-plsql/

Structure of PL/SQL block


```
Declaration Section
BEGIN
Execution Section
EXCEPTION
Exception Section
END;
```


```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
  2      DBMS_OUTPUT.PUT_LINE('Hello World');
  3  END;
  4  /
Hello World

PL/SQL procedure successfully completed.
```

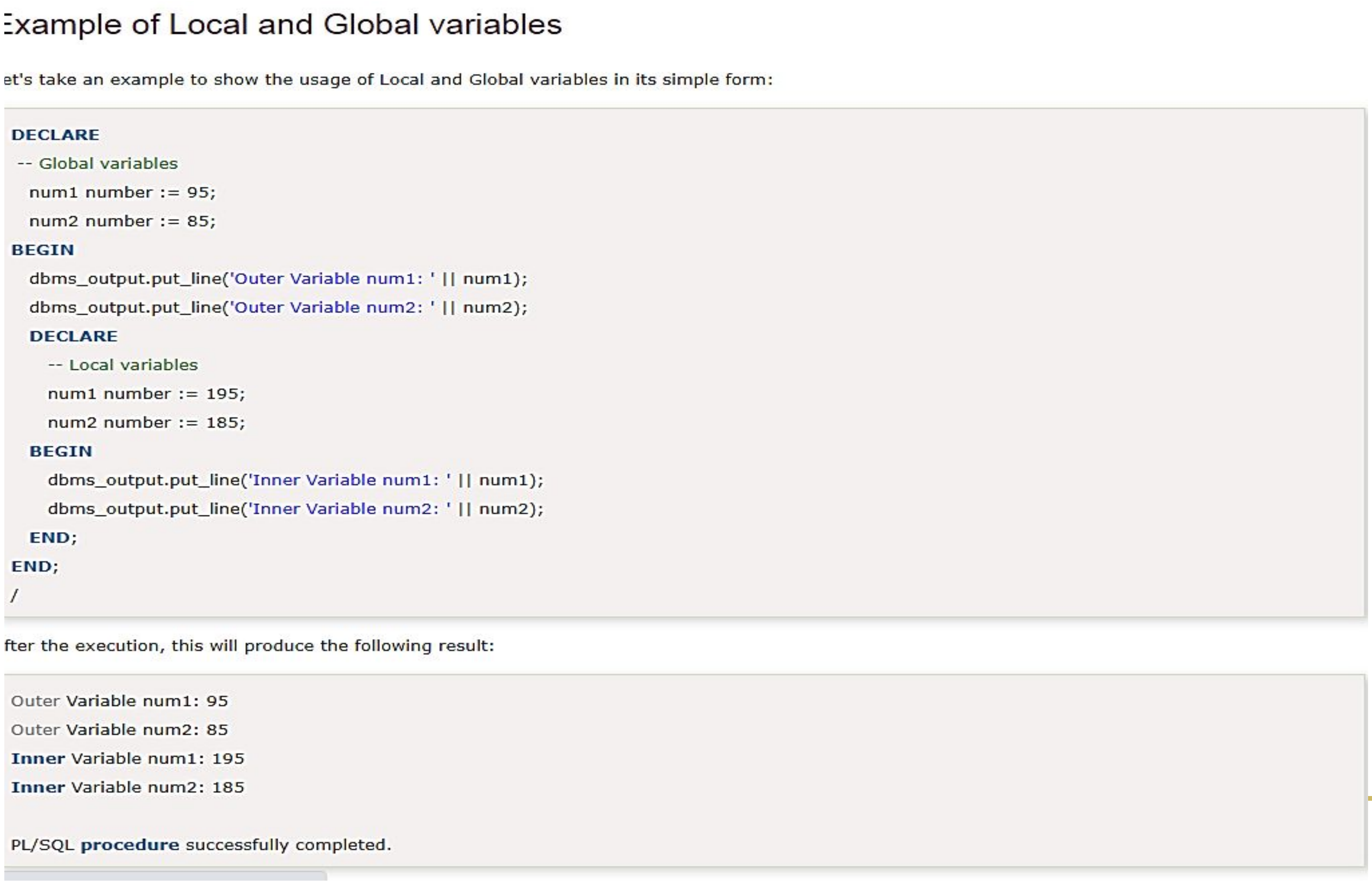
Output:

The Oracle dbms_output is a package that allows us to write data to direct our PL/SQL output to a screen. It has a procedure called put_line that display the information in a line. The package is particularly useful for displaying debugging information

To get input from user at Runtime

```
X:=&X;
```

Variables



Example of Local and Global variables

et's take an example to show the usage of Local and Global variables in its simple form:

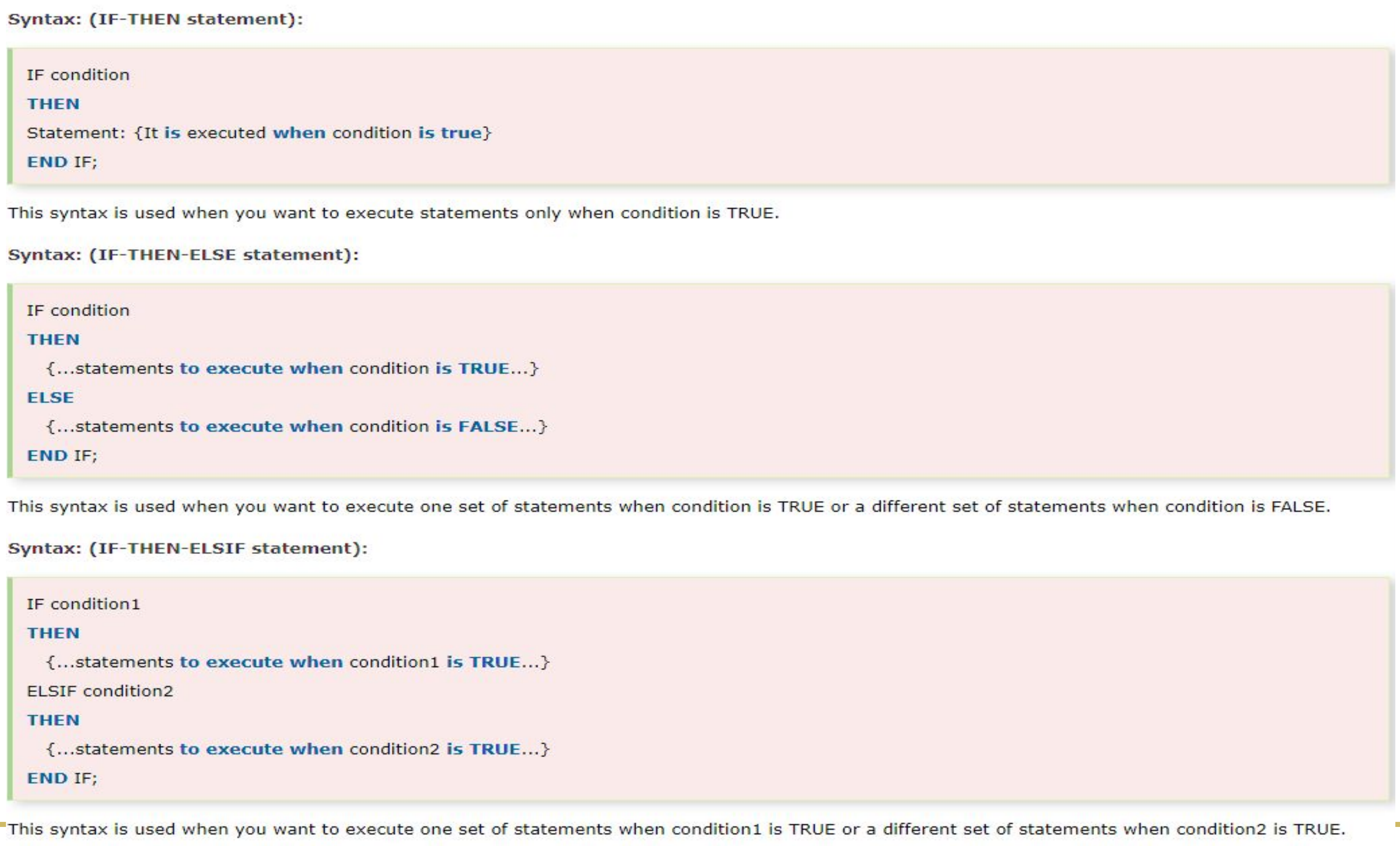
```
DECLARE
 -- Global variables
  num1 number := 95;
  num2 number := 85;
BEGIN
  dbms_output.put_line('Outer Variable num1: ' || num1);
  dbms_output.put_line('Outer Variable num2: ' || num2);
  DECLARE
    -- Local variables
    num1 number := 195;
    num2 number := 185;
  BEGIN
    dbms_output.put_line('Inner Variable num1: ' || num1);
    dbms_output.put_line('Inner Variable num2: ' || num2);
  END;
END;
/
```

fter the execution, this will produce the following result:

```
Outer Variable num1: 95
Outer Variable num2: 85
Inner Variable num1: 195
Inner Variable num2: 185

PL/SQL procedure successfully completed.
```

If-else Statements



Syntax: (IF-THEN statement):

```
IF condition
THEN
Statement: {It is executed when condition is true}
END IF;
```

This syntax is used when you want to execute statements only when condition is TRUE.

Syntax: (IF-THEN-ELSE statement):

```
IF condition
THEN
   {...statements to execute when condition is TRUE...}
ELSE
   {...statements to execute when condition is FALSE...}
END IF;
```

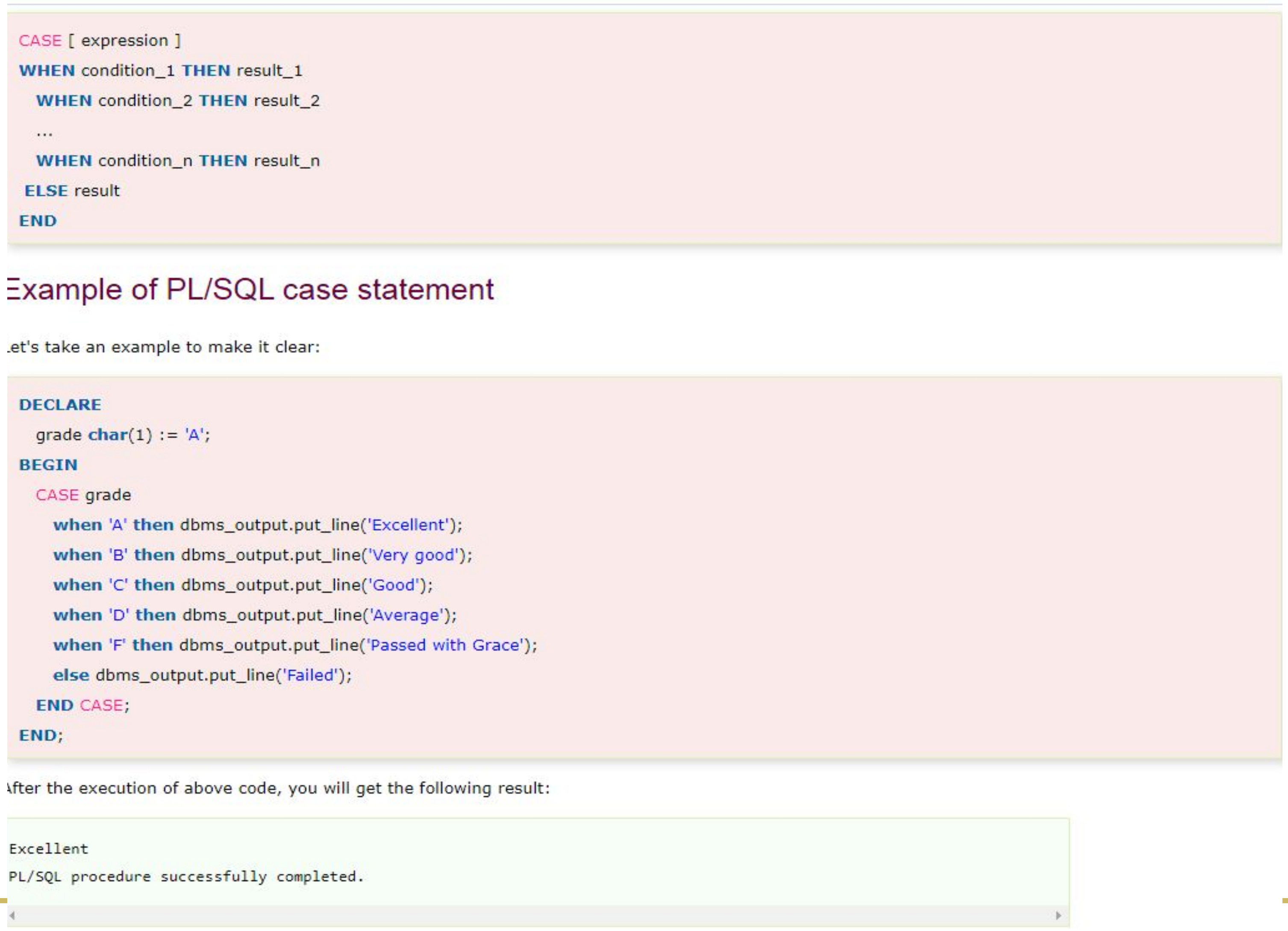
This syntax is used when you want to execute one set of statements when condition is TRUE or a different set of statements when condition is FALSE.

Syntax: (IF-THEN-ELSIF statement):

```
IF condition1
THEN
   {...statements to execute when condition1 is TRUE...}
ELSIF condition2
THEN
   {...statements to execute when condition2 is TRUE...}
END IF;
```

This syntax is used when you want to execute one set of statements when condition1 is TRUE or a different set of statements when condition2 is TRUE.

Case Statement



```
CASE [ expression ]
WHEN condition_1 THEN result_1
  WHEN condition_2 THEN result_2
  ...
  WHEN condition_n THEN result_n
 ELSE result
END
```

Example of PL/SQL case statement

et's take an example to make it clear:

```
DECLARE
  grade char(1) := 'A';
BEGIN
  CASE grade
    when 'A' then dbms_output.put_line('Excellent');
    when 'B' then dbms_output.put_line('Very good');
    when 'C' then dbms_output.put_line('Good');
    when 'D' then dbms_output.put_line('Average');
    when 'F' then dbms_output.put_line('Passed with Grace');
    else dbms_output.put_line('Failed');
  END CASE;
END;
```

fter the execution of above code, you will get the following result:

```
Excellent
PL/SQL procedure successfully completed.
```

Loops & Procedures

```
WHILE <condition>
 LOOP statements;
END LOOP;
```

```
FOR counter IN initial_value .. final_value LOOP
  LOOP statements;
END LOOP;
```

```
CREATE [OR REPLACE] PROCEDURE procedure_name
    [ (parameter [,parameter]) ]
IS
    [declaration_section]
BEGIN
    executable_section
[EXCEPTION
    exception_section]
END [procedure_name];
```

Example

```
    create table user(id number(10) primary key,name varchar2(100));


    create or replace procedure "INSERTUSER"
    (id IN NUMBER, name IN VARCHAR2)
    is
    begin
    insert into user values(id,name);
    end;
    /


BEGIN
   insertuser(101,'Rahul');
   dbms_output.put_line('record inserted successfully');
END;
/
```

PL/SQL Function

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

```
CREATE [OR REPLACE] FUNCTION function_name
(parameter_list) RETURN return_type IS{IS | AS}
BEGIN
  < function_body >
END;
```

PL/SQL Function

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

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