Unit – III

SQL and PL/SQL

- Sub queries
- Joins and its types
- Set operations
- Database objects: View, Index, Sequence, Synonym etc.
- PL/SQL – introduction and its features
- PL/SQL block structure
- Control structures

JOINS

- SQL **Join** is used to fetch data from two or more table, which is joined to appear as single set of data

- Tables are joined on columns that have same **data type** and **data width** in the tables.

- **Types of Joins**
  - INNER JOIN (EQUI JOIN)
  - OUTER JOIN (LEFT , RIGHT, FULL)
  - CROSS JOIN

Join query in two styles

- ANSI-Style
- Theta-Style

<u>INNER JOINS</u>

It returns rows from two or more tables that satisfy the condition.

It compares common columns of tables with = operator.

<u>ANSI-style</u>

SELECT

{ *|*column_name1*, *column_name2, .. column_nameN }*

FROM

<Table-Name 1> **INNER JOIN** *< Table-name 2>*

ON

<Table-Name 1> **.** *<ColumnName1>* **=** *<Table-Name 2>* **.** *<ColumnName2>*

[**Where** conditions]

[**ORDER BY**

{ column_name1.., column_nameN [**ASC / DESC**] **}**] **;**

- *ColumnName1 in TableName1 is usually that table's Primary Key.*
- *ColumnName2 in TableName2 is a foreign key in that table*
- *ColumnName1 and ColumnName2 must have the same data type & size.*

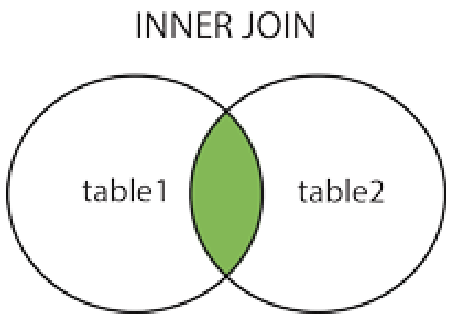
INNER JOIN Example

Without Join

- SELECT Client_Name , Bill_No, Bill_Amt
  FROM Client_master , bill_details

With Join

- SELECT Client_Name , Bill_No, Bill_Amt
  FROM Client_master **INNER JOIN** bill_details
  ON Client_master**.**client_no = bill_details**.**client_no
- SELECT Client_Name , Bill_No, Bill_Amt
  FROM bill_details **INNER JOIN** Client_master
  ON Client_master**.**client_no = bill_details**.**client_no
- SELECT Client_Name , Bill_No, Bill_Amt
  FROM bill_details **INNER JOIN** Client_master
  ON bill_details**.**client_no = Client_master**.**client_no
- SELECT C**.**Client_Name , B**.**Bill_No, B**.**Bill_Amt
  FROM Client_master C **INNER JOIN** bill_details B
  ON C**.**client_no = B**.**client_no



INNER JOINS

Theta-style

SELECT

{ * | *column_name1, column_name2, .. column_nameN }*

FROM

<Table-Name 1> **,** *< Table-name 2>*

Where

<Table-Name 1> **.** *<ColumnName1>* **=** *<Table-Name 2>* **.** *<ColumnName2>*

AND *<Condition>*

[**ORDER BY**

{ column_name1.., column_nameN [**ASC / DESC**] **}**] **;**

- *ColumnName1 in TableName1 is usually that table's Primary Key.*
- *ColumnName2 in TableName2 is a foreign key in that table*
- *ColumnName1 and ColumnName2 must have the same data type and the same size.*

INNER JOIN Example

- SELECT Client_Name , Bill_No, Bill_Amt
  FROM Client_master , bill_details
  WHERE Client_master**.**client_no = bill_details**.**client_no
- SELECT Client_Name , Bill_No, Bill_Amt
  FROM bill_details **,** Client_master
  WHERE Client_master**.**client_no = bill_details**.**client_no
- SELECT Client_Name , Bill_No, Bill_Amt
  FROM bill_details **,** Client_master
  WHERE bill_details**.**client_no = Client_master**.**client_no
- SELECT C**.**Client_Name , B**.**Bill_No, B**.**Bill_Amt
  FROM Client_master C **,** bill_details B
  WHERE C**.**client_no = B**.**client_no

INNER JOIN Example

- **E.g.**
- CUST_MST (Cust_No, Fname, Mname, Lname,)
- ADDR_DET (Code_No, Addr1, Addr2, City, Pincode)
- **List the customers along with their multiple address details.**
- SELECT C.Cust_No, (C.Fname || ' ' || C.Mname || ' ' || C.Lname) "Customer", (A.Addr1 || ' ' || A.Addr2 || ' ' || A.city || ' ' || A.Pincode) "Address" FROM CUST_MST C , ADDR_DET A

  WHERE C.Cust_No = A.Code_No

- **List the customers along with their multiple address details whose name start with 'A'.**
- SELECT C.Cust_No, (C.Fname || ' ' || C.Mname || ' ' || C.Lname) "Customer", (A.Addr1 || ' ' || A.Addr2 || ' ' || A.city || ' ' || A.Pincode) "Address" FROM CUST_MST C , ADDR_DET A

  WHERE C.Cust_No = A.Code_No **AND** C.Fname like 'A%'

OUTER JOIN Example

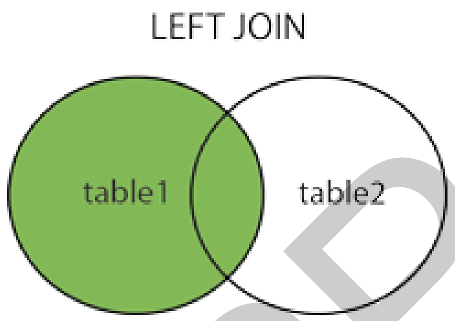
- It returns rows satisfy the conditions and also returns rows from one of the joining tables which did not satisfy the condition.

- The table that is chosen for this “bypass” of conditional requirements is determined by the directionality or “side” of the join, typically referred to as LEFT or RIGHT outer joins.

- The other table values should be display as NULL values as a part of joining condition.

OUTER JOIN Example

Left Outer Join

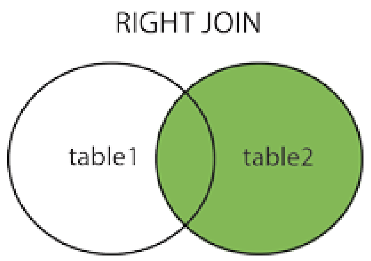


- **E.g. List the employee details along with the contact details (if any) Using Left Outer Join.**
- SELECT C.Cust_No, (C.Fname || ' ' || C.Mname || ' ' || C.Lname) "Customer", (A.Addr1 || ' ' || A.Addr2 || ' ' || A.city || ' ' || A.Pincode) "Address" FROM CUST_MST C **LEFT OUTER JOIN** ADDR_DET A

  ON C.Cust_No = A.Code_No
- SELECT C.Cust_No, (C.Fname || ' ' || C.Mname || ' ' || C.Lname) "Customer", (A.Addr1 || ' ' || A.Addr2 || ' ' || A.city || ' ' || A.Pincode) "Address" FROM CUST_MST C **,** ADDR_DET A

  WHERE C.Cust_No = A.Code_No **(+)**

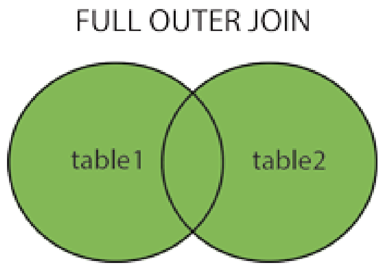
OUTER JOIN Example



- **Right Outer Join**

- **List the employee details along with the contact details (if any) Using Right Outer Join**
- SELECT C.Cust_No, (C.Fname || ' ' || C.Mname || ' ' || C.Lname) "Customer", (A.Addr1 || ' ' || A.Addr2 || ' ' || A.city || ' ' || A.Pincode) "Address" FROM CUST_MST C **RIGHT OUTER JOIN** ADDR_DET A **ON** C.Cust_No = A.Code_No
- SELECT C.Cust_No, (C.Fname || ' ' || C.Mname || ' ' || C.Lname) "Customer", (A.Addr1 || ' ' || A.Addr2 || ' ' || A.city || ' ' || A.Pincode) "Address" FROM ADDR_DET A **,** CUST_MST C **ON** A.Code_No**(+)** = C.Cust_No
- SELECT C.Cust_No, (C.Fname || ' ' || C.Mname || ' ' || C.Lname) "Customer", (A.Addr1 || ' ' || A.Addr2 || ' ' || A.city || ' ' || A.Pincode) "Address" FROM ADDR_DET A **,** CUST_MST C **ON** C.Cust_No = A.Code_No**(+)**

OUTER JOIN Example



Full Outer Join

- **Full Outer Join r**eturns all the rows from the left table (Customers), and all the rows from the right table (Address_Detail).
- If there are rows in "Customers" that do not have matches in "Address_Detail", or
- if there are rows in "Address_Detail" that do not have matches in "Customers", those rows will be listed as well.
- SELECT C.Cust_No, (C.Fname || ' ' || C.Mname || ' ' || C.Lname) "Customer", (A.Addr1 || ' ' || A.Addr2 || ' ' || A.city || ' ' || A.Pincode) "Address" FROM CUST_MST C **FULL OUTER JOIN** ADDR_DET A **ON** C.Cust_No = A.Code_No
- SELECT C.Cust_No, (C.Fname || ' ' || C.Mname || ' ' || C.Lname) "Customer", (A.Addr1 || ' ' || A.Addr2 || ' ' || A.city || ' ' || A.Pincode) "Address" FROM ADDR_DET A **,** CUST_MST C **ON** A.Code_No**(+)** = C.Cust_No**(+)**

CROSS JOIN Example

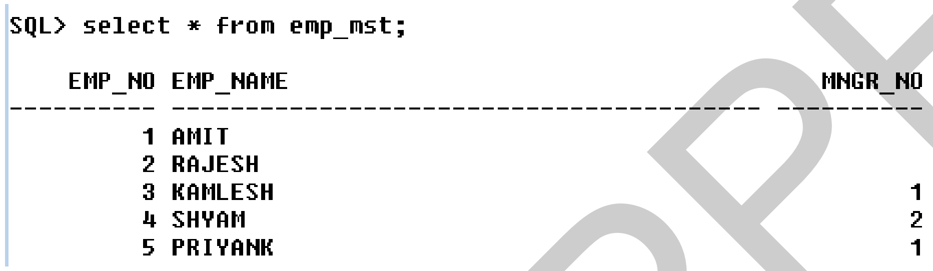
- The Cross Join clause produces the cross-product of two tables.
- A cross join or Cartesian product is formed when every row from one table is joined to all rows in another.
- Suppose the source and target tables have five and four rows, respectively, a cross join between them results in (5 * 4 = 20) rows being returned.
- Sales_Org (Org_id, Org_name)
  - Domestic, International
- Sales_Channel (Channel_id, Channel_name) Wholesale etc.
  - E-commerce, TV Shopping, Warehouse

- **SELECT** Org_name , Channel_name
  FROM Sales_Org **CROSS JOIN** Sales_Channel

- **SELECT** Org_name , Channel_name
  FROM Sales_Org , Sales_Channel

SELF JOIN Example

- Join a table to itself is known as self join.
- Two rows from the same table combine to form a result row.
- Two copies of the same table have to be opened in memory, so each table is opened using an alias.

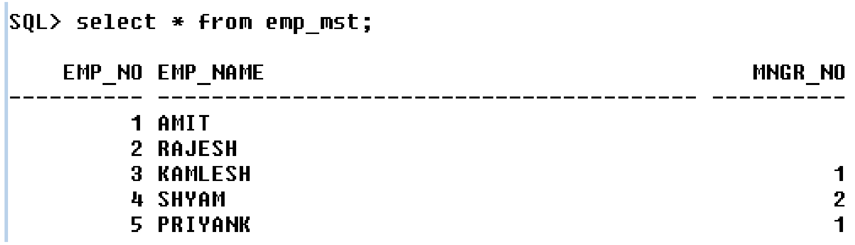


```
SQL> select * from emp_mst;

    EMP_NO EMP_NAME                                    MNGR_NO
---------- ---------------------------------------- ----------
         1 AMIT
         2 RAJESH
         3 KAMLESH                                           1
         4 SHYAM                                             2
         5 PRIYANK                                           1
```

EMP_MST **(E)**

EMP_MST **(M)**



```
SQL> select * from emp_mst;

    EMP_NO EMP_NAME                                    MNGR_NO
---------- ---------------------------------------- ----------
         1 AMIT
         2 RAJESH
         3 KAMLESH                                           1
         4 SHYAM                                             2
         5 PRIYANK                                           1
```

- **SELECT** E.Emp_Name “Employee” , M.Emp_Name “Manager”
  FROM EMP_MST E , EMP_MST M
  WHERE E.Mngr_No = M.Emp_No

Examples of SelfJoin

- Display name of categories and its parent category.
- Display menu and submenu items.
- Display various combination of colors and shades.

Subqueries

- It also known as nested query. It means SQL statements inside another SQL statement.
- It can be used for the following.
  - To create backup tables with data
    - **CREATE** TABLE TEMP_BAK AS (SELECT * FROM TEMP);
  - To insert records in a target table.
    - **INSERT** INTO TEMP_BAK (SELECT * FROM TEMP);
  - To update record in a target table
    - **UPDATE** temp_bak SET d = (SELECT d FROM temp WHERE rownum = 1);
  - To delete a record in a target table
    - **DELETE** FROM temp_bak WHERE d = (SELECT d FROM temp WHERE rownum = 1);
  - To create a view
    - **CREATE** VIEW V_TEMP AS SELECT * FROM TEMP;

Subqueries

E.g.

CUST_MST (Cust_No, Fname, Mname, Lname,)

ADDR_DET (Code_No, Addr1, Addr2, City, Pincode)

- **Retrieve the address of a customer named ‘Amit Patel'**
  - **SELECT** Code_No, Addr1 || ' ' || Addr2 || ' ' || city || ' ' || Pincode "Address"

    FROM ADDR_DET

    WHERE Code_No = **(SELECT** Cust_No **FROM** CUST_MST **WHERE** Fname = 'AMIT' **AND** Lname='PATEL'**)**

Subqueries

CUST_MST (Cust_No, Fname, Mname, Lname,)

ADDR_DET (Code_No, Addr1, Addr2, City, Pincode)

- **Find the customers who are staying in 'ANAND'**
  - **SELECT** Cust_No, Fname, Mname, Lname
    FROM Cust_Mst
    WHERE Cust_No **IN** (**SELECT DISTINCT** Code_No
    FROM ADDR_DET
    WHERE city = 'ANAND')

Set Operation : Union

- It is used to combine the results of two or more SELECT statements without returning any duplicate rows.

- Each SELECT statement must have
  - The same number of columns selected
  - The same number of column expressions
  - The same data type and
  - Have them in the same order

- **Syntax**

SELECT column1 [, column 2]
FROM table-name1 [, table-name 2]
[WHERE Condition]
UNION
SELECT column1 [, column 2]
FROM table-name1 [, table-name 2]
[WHERE Condition]

```
CREATE TABLE Customers( CustomerID VARCHAR2(5) PRIMARY KEY,
            CustomerName VARCHAR2(40),
                                    Contact_No NUMBER(10),
                                    Address VARCHAR2(40),
                                    City VARCHAR2(40),
                                    PostalCode NUMBER(6),
                                    Country VARCHAR2(40))

INSERT INTO Customers( CustomerID, CustomerName, City, Country) VALUES ( '&CustomerID', '&CustomerName' ,
'&City', '&Country')

CREATE TABLE Suppliers(   SupplierID VARCHAR2(5) PRIMARY KEY,
                                    SupplierName VARCHAR2(40),
                                    Contact_No NUMBER(10),
                                    Address VARCHAR2(40),
                                    City VARCHAR2(40),
                                    PostalCode NUMBER(6),
                                    Country VARCHAR2(40))

INSERT INTO Suppliers( SupplierID, SupplierName, City, Country) VALUES ( '&SupplierID', '&SupplierName' , '&City',
'&Country')
```

Set Operation : Union

- The SQL UNION operator is used to return the results of 2 or more SELECT statements.
- If a record exists in any query , it will be part of UNION results.

E.g.

- Customers(CustomerID, CustomerName, Contact_No, Address, City, PostalCode, Country)
- Suppliers(SupplierID, SupplierName, Contact_No, Address, City, PostalCode, Country)

- Display all cities (without duplicate values) from "Customers" and "Suppliers":

  SELECT city FROM customers **UNION** SELECT city FROM Suppliers ORDER BY City

- Display all cities (duplicate values also) from "Customers" and "Suppliers":

  SELECT city FROM customers **UNION ALL** SELECT city FROM Suppliers ORDER BY City

Set Operation : Intersect

- It is used to combine the results of two or more SELECT statements but returns rows only from the first SELECT statement that are match to a row in the second SELECT statement.

- **Syntax**

```
SELECT column1 [, column 2]
FROM table-name1 [, table-name 2 ]
[WHERE Condition ]
INTERSECT
SELECT column1 [, column 2]
FROM table-name1 [, table-name 2 ]
[WHERE Condition ]
```

Set Operation : Intersect

- **E.g.**
- Cust_Mst(CustomerID, CustomerName, ContactName, Address, City, PostalCode, Country)
- Supp_Mst(SupplierID, SupplierName, SupplierName, Address, City, PostalCode, Country)

- Display common cities from "Customers" and "Suppliers":

  SELECT City FROM Customers **INTERSECT** SELECT City FROM Suppliers ORDER BY City;

Set Operation : MINUS

- The SQL MINUS operator is used to return all rows in the first SELECT statement that are not returned by the second SELECT statement.

- **Syntax**

```
SELECT column1 [, column 2]
FROM table-name1 [, table-name 2 ]
[WHERE Condition ]
MINUS
SELECT column1 [, column 2]
FROM table-name1 [, table-name 2 ]
[WHERE Condition ]
```

Set Operation : Minus

- **E.g.**
- Cust_Mst(CustomerID, CustomerName, ContactName, Address, City, PostalCode, Country)
- Supp_Mst(SupplierID, SupplierName, SupplierName, Address, City, PostalCode, Country)

- Display cities from "Customers“ which are not in "Suppliers":

  SELECT City FROM Customers **MINUS** SELECT City FROM Suppliers ORDER BY City;

Views

- **What is view?**
  - An VIEW is a virtual table that does not physically exist. It is created by a query with single table or joining multiple tables.
- **Why view is created?**
  - Views have long been used to hide the tables that actually contain the data you are querying. Also, views can be used to restrict the columns that a given user has access to.

View

- It is a logical representation of a table or combination of tables. In reality, a view is a stored query.
- A view derives its data from the tables on which it is based.
- All operations performed on a view actually affect the base table of the view.
- It is always constructed at runtime
- A view is useful for hiding "sensitive" data columns.
- **Syntax**

  CREATE VIEW view-Name

  [(Simple-column-Name [, Simple-column-Name] *)]

  AS SELECT Statement

E.g.

```
CREATE VIEW v_empdata AS SELECT  eno, ename FROM emp;
```

SEQUENCES

- Oracle provides the capability to generate sequences of Unique numbers, and they are called sequences.
- Sequences are used to generate unique primary keys automatically.

Syntax :

```
CREATE SEQUENCE <sequence name>
    [INCREMENT BY <number>]
    [START WITH <start value number>]
    [MAXVALUE <MAXIMUM VLAUE NUMBER>];
```

- A sequence is referenced in SQL statements with the NEXTVAL and CURRVAL pseudocolumns.
- Each new sequence number is generated by a reference to the sequence pseudocolumn NEXTVAL
- current sequence number can be referenced using the pseudo-column CURRVAL.

Example

> CREATE SEQUENCE **Emp_Seq**

INCREMENT BY 1

START WITH 1

MAXVALUE 1000

To find the current val of seq. :: **SELECT <Sequence Name >.CurrVal FROM DUAL;**

> SELECT EMP_SEQ.CURRVAL FROM DUAL;

To find the next val of seq. :: **SELECT <Sequence Name > . NextVal FROM DUAL;**

> SELECT EMP_SEQ.NEXTVAL FROM DUAL;

To insert record in table using sequence

> CREATE TABLE Employee_Temp(Eno NUMBER(3) PRIMARY KEY, Ename VARCHAR2(40));

> INSERT INTO Employee_temp values (Emp_Seq.NextVal,'ALPESH');

> SELECT * FROM EMPLOYEE_TEMP;

To remove sequence :: DROP SEQUENCE **<Sequence Name >**

DROP SEQUENCE EMP_SEQ;

Index

- It is a structure associated with tables that allow SQL queries to execute more quickly against a table.
- Indexes are logically and physically independent of the data in the associated table. It require storage space.
- Index is created on columns which are frequently retrieve from the table.
- The database automatically maintains indexes when you insert, update, and delete rows of the associated table.
- By default Indexes are created for columns in Unique, primary key, and foreign key constraints.
- **Syntax**

CREATE INDEX index-Name

ON table-Name (Simple-column-Name [ASC | DESC]

[, Simple-column-Name [ASC | DESC]] *)

E.g.

```
CREATE INDEX I_Name ON emp (ename ASC);
```

SYNONYMS

- It is an alternative name of a table.
- It can provide a level of security by masking the name and owner of an object.
  - **Syntax**

    CREATE SYNONYM synonym-name **FOR {** table-name | view-name **}**
- **E.g. CREATE SYNONYM** s_emp **FOR** emp**;**

- A synonym can be used instead of original table in SELECT , INSERT , DELETE & UPDATE Statements.

- **To DROP SYNONYM :**
  - **Syntax**

    DROP SYNONYM synonym-name

  E.g.

    DROP SYNONYM s_emp;

Difference between SQL and PL / SQL

- SQL : Structured Query Language
- PL/SQL : Procedural Language / Structured Query Language

- SQL is used to write queries, DDL and DML statements.
- PL/SQL is used to write program blocks, functions, procedures triggers, and packages.

- SQL is executed one statement at a time.
- PL/SQL is executed as a block of code.

- SQL is declarative, i.e., it tells the database what to do but not how to do it.
- PL/SQL is procedural, i.e., it tells the database how to do things.

- SQL create more network traffic .
- PL/SQL create less network traffic.

- SQL does not support error handling
- PL/SQL supports error handling

- SQL can be embedded within a PL/SQL program.
- PL/SQL can't be embedded within a SQL statement.

Features of PL / SQL

- Block structure :

  A block is a unit of code that provides execution and scoping boundaries.

- Variable and Data Types
- Control Structure and loops
- Error Handling
- Procedures and functions
- Packages
- Cursor

Block Structure

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

Simple Block (Display Output On Screen)

Package : DBMS_OUTPUT : To display output on screen.

In-built function to print the message.

Functions : PUT : To display the content on current line &
PUT_LINE : To display the content on next line.

For Single line comment: --   **For Multiline comment:** /* ….. */

```
BEGIN
    DBMS_OUTPUT.put('Apple');      DBMS_OUTPUT.put('    ');
    DBMS_OUTPUT.put('Banana');   DBMS_OUTPUT.put_line('');

    DBMS_OUTPUT.put_line('Pinaple');
    DBMS_OUTPUT.put_line('First PL/SQL Program');
 -- DBMS_OUTPUT.put_line('Animals');            Single Line Comment
    /*     DBMS_OUTPUT.put_line('Lion');       Multi line Comment
           DBMS_OUTPUT.put_line('Tiger');
    */
END;
/
```

Input & Output Of Variable Data

```
DECLARE
    v_no  NUMBER(2);

BEGIN
   v_no := &Number1;
   DBMS_OUTPUT.put_line('Value of v_no :: ' || v_no);
END;
/
```

PL/SQL Block : Select Statement With INTO Clause

EMP_M (emp_no NUMBER(2)   PRIMARY KEY, emp_name VARCHAR2(50))

Display the employee no. and name for given emp_no.

```
DECLARE
        v_no                    NUMBER;
        v_name      VARCHAR2(40);
BEGIN
    v_no := &v_no;

   SELECT emp_name INTO v_name  FROM emp_m WHERE emp_no = v_no ;

   DBMS_OUTPUT.put_line('Employee No. :'|| v_no);
   DBMS_OUTPUT.put_line('Employee Name :'|| v_name);
END;
```

%TYPE attribute in variable declaration

It is used to declare a field with the same data type as -- that of a specified table's column:

```
DECLARE
   v_no  emp_m.emp_no%TYPE ;
   v_name emp_m.emp_name%TYPE;
BEGIN
   v_no := &v_no;

   SELECT emp_name INTO v_name FROM emp_m WHERE emp_no = v_no;

   DBMS_OUTPUT.PUT_LINE('Employee Name ::' || v_name);
 END;
/
```

%ROWTYPE attribute for variable declaration

Program: Display the employee details for given employee no.

```
DECLARE
   v_emprec  emp_master%ROWTYPE;
   v_empno   emp_master.emp_no%TYPE ;
BEGIN
    v_empno:=&Employee_No ;
    SELECT * INTO v_emprec FROM emp_master WHERE emp_no = v_empno ;

    DBMS_OUTPUT.put_line ('Employee Details');
    DBMS_OUTPUT.put_line ('-----------------');
    DBMS_OUTPUT.put_line ('Employee No.  :: ' || v_emprec.emp_no);
    DBMS_OUTPUT.put_line ('Employee Name  :: '|| v_emprec.emp_name);
    DBMS_OUTPUT.put_line ('Employee dept.  :: '|| v_emprec.dept);
    DBMS_OUTPUT.put_line ('Employee desig.  :: '|| v_emprec.desig);
END;
/
```

Control Structure

- It determines the order in which statements are executed in a block.
- It tests a condition, then executes sequence of statements.
- A *condition* returns a Boolean value (TRUE or FALSE).
- **Control structures** in the PL / SQL:
  1. **Conditional**
  2. **Iterative**

Control Structure

1. **Conditional**

The sequence of statements are executed or not depends on the value of a condition.

- **Three forms of IF statements:**

  IF-THEN,

  IF-THEN-ELSE, and

  IF-THEN-ELSIF.

- **CASE statement:** Evaluate a single condition & many alternative actions.

2. **Iterative:**
   - To execute a sequence of statements multiple times.
   - There are three forms of **LOOP statements:**
     1. **LOOP**
     2. **WHILE-LOOP**
     3. **FOR-LOOP.**

Conditional Control Structure

Simple If

```
IF condition THEN
    sequence_of_statements
END IF;
```

IF – Else Statement

```
IF condition THEN
    sequence_of_statements1
ELSE
    sequence_of_statements2
END IF;
```

Conditional Control Structure

Example : To Find The Entered Number Is Even Or Odd

```
DECLARE
    v_no   NUMBER (3);
BEGIN
    v_no:=&Number;
    IF ( MOD(v_no,2) = 0  ) THEN
           DBMS_OUTPUT.PUT_LINE( ' Entered No. is Even No.');
    ELSE
           DBMS_OUTPUT.PUT_LINE( ' Entered No. is Odd No.');
    END IF;
END;
/
```

Conditional Control Structure

IF-THEN-ELSIF Statement

```
IF condition1 THEN
    sequence_of_statements1
ELSIF condition2 THEN
    sequence_of_statements2
ELSE
    sequence_of_statements3
END  IF;
```

Conditional Control Structure

Example : To Find Maximum Number Out Of Three Number

```
DECLARE
    v_no1   NUMBER (3); v_no2   NUMBER (3); v_no3   NUMBER (3);
BEGIN
    v_no1 := &Number1;  v_no2 := &Number2;  v_no3 := &Number3;
    IF ( (v_no1 > v_no2)  AND (v_no1 > v_no3) ) THEN
          DBMS_OUTPUT.PUT_LINE( ' Maximum No. ::' || v_no1);
    ELSIF ( (v_no2 > v_no1)  AND (v_no2 > v_no3) ) THEN
          DBMS_OUTPUT.PUT_LINE( ' Maximum No. ::' || v_no2);
    ELSE
          DBMS_OUTPUT.PUT_LINE( 'Maximum No. ::' || v_no3);
    END IF;
END;
 /
```

Conditional Control Structure

```
CASE [ expression ]

    WHEN condition_1 THEN sequence_of_statements1;

    WHEN condition_2 THEN sequence_of_statements2;
    ………………….
    ………………….
    ………………….

    WHEN condition_N THEN sequence_of_statementsN;

    [ELSE sequence_of_statementsN+1;]

END CASE;
```

Conditional Control Structure

Example : Enter the grade and find corresponding message.

```
DECLARE
    grade CHAR(1);
BEGIN
    grade := '&Grade';
    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('Fair');
            WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
            ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
    END CASE;
END;
```

Iterative Control Structure

Simple Loop :

```
LOOP
        sequence_of_statements
END LOOP;
```

Example : Find the sum of first N numbers.

```
DECLARE
    v_no number(3);     inc number(3) := 1;     ans number(3) := 0;
BEGIN
    v_no:=&Number;
    LOOP
        ans := ans + inc;
        inc := inc + 1 ;
        EXIT WHEN (inc > v_no);
    END LOOP;
    dbms_output.put_line('Sum of first  ' || v_no || '  Numbers::' || ans);
END;
```

Iterative Control Structure

While Loop : **WHILE** condition

```
WHILE condition
LOOP
   sequence_of_statements
END LOOP;
```

Example : Find the sum of first N numbers.

```
DECLARE
    v_no number(3);        inc number(3) := 1;         ans number(3) := 0;
BEGIN
    v_no := &Enter_No;
    WHILE ( inc  <=  v_no)
    LOOP
          ans := ans + inc;
          inc := inc + 1 ;
    END LOOP;
    dbms_output.put_line('Sum of first   ' || v_no || '  numbers ::' || ans);
END;
```

Iterative Control Structure

For Loop

```
FOR counter  IN  [REVERSE] initial_value .. final_value
LOOP
   sequence_of_statements;
END LOOP;
```

Example : Find the sum of first N numbers.

```
DECLARE
    v_no number(3);      ans number(3) := 0;
BEGIN
    v_no := &Enter_No;
    FOR  inc IN 1 .. v_no
    LOOP
           ans := ans + inc;
    END LOOP;
    dbms_output.put_line('Sum of first  ' || v_no || ' Numbers::' || ans);
END;
```

Write a PL/SQL blocks for following programs

1. Write a PL/SQL block to find the entered no. is ever or odd.
2. Write a PL/SQL block to accept two numbers and display its addition, subtraction, multiplication and division.
3. Write a PL/SQL block to find minimum & maximum value from entered three numbers.
4. Write a PL/SQL block to find the sum of first N numbers.
5. Write a PL/SQL block to find the factorial of a given number.
6. Write a PL/SQL block to find sum of Even numbers and Odd numbers between entered two numbers.
7. Consider the Client_Master (Client_Id, Client_Name) table with Client_Id as primary key.
   a) Write a block to accept the Client_Id and Client_Name from user and store it into table.
   b) Write a block to accept the Client_Id from user and display its name in upper case.
   c) Write a block to delete a record form Client_Master for given Client_Id.
   d) Write a block to change the contact no. to 9999911111 for given Client_Id