1. **Create structured data types of ORDBMS and perform operations- create table using structured data types, insert data and solve queries.**

CREATE TABLE Employees (

EmployeeID INT,

Name VARCHAR(50),

Age INT,

Department VARCHAR(50),

Salary FLOAT

);

INSERT INTO Employees (EmployeeID, Name, Age, Department, Salary)

VALUES

(1, 'John Doe', 30, 'Sales', 50000),

(2, 'Jane Smith', 35, 'Marketing', 60000),

(3, 'Robert Johnson', 45, 'Finance', 70000),

(4, 'Emily Davis', 28, 'HR', 45000);

SELECT Name, Salary FROM Employees;

SELECT \* FROM Employees WHERE Age > 30;

SELECT AVG(Salary) FROM Employees WHERE Department = 'Sales';

Note : if this inserting methods isn’t working then insert records one by one.

**Related Practical Lab code.**

create type student11 as object (name varchar(20), roll int, dept varchar(20));

desc student11;

drop type student11;

create table stdnt of student11(CONSTRAINT c1 PRIMARY KEY (roll));

insert into stdnt(name, roll, dept) values('abc',121,'CSE');

insert into stdnt(name, roll, dept) values('pqr',122,'CSE');

insert into stdnt(name, roll, dept) values('xyz',123,'MECH');

insert into stdnt values('xys',124,'MECH');

select \* from stdnt;

select \* from stdnt where roll=122;

select p.name from stdnt p where p.roll=121;

select \* from stdnt p where p.name='abc';

update stdnt p SET p.name='abcd' where p.roll=121;

select \* from stdnt;

create table student21 of student11(CONSTRAINT c2 PRIMARY KEY (roll));

insert into student21 select \* from stdnt where roll=121;

select \* from student21;

INSERT INTO Table3

(name, roll, dept)

SELECT name,roll,dept

FROM stdnt CROSS JOIN student21;

create table mixed as(select stdnt.name, stdnt.roll, student21.dept from stdnt, student21 where stdnt.roll = student21.roll );

select \* from mixed;

1. **Implement different type of Inheritance in ORDBMS.**

In Object-Relational Database Management Systems (ORDBMS), multiple multilevel hierarchical inheritances are not natively supported. ORDBMS typically support single-level inheritance, where a derived class extends a single base class. However, it's possible to achieve multiple levels of inheritance by applying the concept recursively.

--university under science commerce arts then dept and student inheritence.

create type university as object(roll int)NOT FINAL;

create type science UNDER university(std\_name varchar(20), department varchar(20));

create type commerce UNDER university(std\_name varchar(20), department varchar(20));

create type arts UNDER university(std\_name varchar(20), department varchar(20));

describe university;

create table info of science;

desc info;

insert into info values(121,'suraj', 'CSE');

insert into info values(122,'suyesh', 'CIVIL');

select \* from info;

create table info1 of science;

desc info1;

insert into info1 values(123,'nikhil', 'MECH');

insert into info1 values(124,'pavan', 'AERO');

select \* from info1;

create table info2 of science;

desc info2;

insert into info2 values(125,'pranav', 'FOOD');

insert into info2 values(126,'satej', 'ENTC');

select \* from info2;

**Related Practical Lab code.**

/\*implementation of inheritance\*/

create type living as object(name varchar(20), type varchar(20), id number)NOT FINAL;

/\*NOT FINAL represents having child\*/

desc living;

create type human UNDER living();

desc human;

/\*human is child of living\*/

create table stdinfo of human;

/\*created table of type human(no need to define attributes of table again)\*/

desc stdinfo;

/\*inserting values in table\*/

insert into stdinfo values('ABC','CSE',3034);

insert into stdinfo values('XYZ','MECH',3033);

insert into stdinfo values('LMN','CIVIL',3032);

select \* from stdinfo;

drop type human;/\*we cannot delete type directly\*/

drop table stdinfo;

drop type human;/\*now we can delete type\*/

/\* college dept TY table rollcalllist\*/

create type college1 as object(clg\_name varchar(20), address varchar(20)) NOT FINAL;/\*college1 is parent\*/

create type dept1 UNDER college1(dept\_name varchar(20), building\_no int) NOT FINAL;/\*dept1 is child of college1\*/

create type TY UNDER dept1(roll int,std\_name varchar(20)); /\*TY is child of dept1\*/

create table rollList of TY;

desc rollList;/\*it has all attributes of college1, dept1 as well as TY\*/

insert into rollList values('ADCET','ASHTA','CSE',4,3034,'Suraj'); /\*inserting values of all attributes\*/

insert into rollList values('RIT','ISLAMPUR','MECH',4,3035,'Ajay');

insert into rolllist(roll, std\_name) values(121,'XZF');/\*inserting values of only some attributes\*/

select \* from rollList;

**3. Implement multivalued attributes complex types in ORDBMS.**

CREATE TYPE Phone\_typ AS OBJECT (

PhoneNumber VARCHAR2(20),

PhoneType VARCHAR2(20)

);

CREATE TYPE Phones\_ntt AS TABLE OF Phone\_typ;

CREATE TABLE Employees (

EmployeeID NUMBER,

Name VARCHAR2(50),

Phones Phones\_ntt

) NESTED TABLE Phones STORE AS Phones\_nt;

INSERT INTO Employees (EmployeeID, Name, Phones)

VALUES (1, 'John Doe', Phones\_ntt(Phone\_typ('1234567890', 'Home'), Phone\_typ('9876543210', 'Work')));

SELECT e.EmployeeID, e.Name, p.PhoneNumber, p.PhoneType

FROM Employees e, TABLE(e.Phones) p;

In this example, we have defined a complex type called **Phone\_typ**, which represents a phone number along with its type. The **Phone\_typ** object has two attributes: **PhoneNumber** and **PhoneType**.

**4. Implement vertical fragmentation in distributed DBMS**

Vertical fragmentation in a distributed database refers to the process of dividing a table's columns into separate fragments, with each fragment stored on a different database node or server. This technique allows for distributing different attributes or columns of a table across multiple nodes to improve performance or achieve data locality.

To perform vertical fragmentation, you can split the table into two fragments: "Fragment1" and "Fragment2", where "Fragment1" contains the EmployeeID, Name, and Department columns, and "Fragment2" contains the Age and Salary columns.

Assuming you have two database nodes, Node1 and Node2, you can create the table fragments as follows:

**On node 1:**

CREATE TABLE Fragment1 ( EmployeeID INT PRIMARY KEY, Name VARCHAR(50), Department VARCHAR(50));

INSERT INTO Fragment1 VALUES(1, 'John Doe', 'Sales');

INSERT INTO Fragment1 VALUES(2, 'Doe Lipa', 'Marketing');

**On node 2:**

CREATE TABLE Fragment2 (EmployeeID INT PRIMARY KEY, Age INT, Salary FLOAT);

INSERT INTO Fragment2 VALUES(1, 30, 50000);

INSERT INTO Fragment2 VALUES(2, 35, 60000);

SELECT F1.EmployeeID, F1.Name, F2.Age, F1.Department, F2.Salary

FROM Fragment1 F1

JOIN Fragment2 F2 ON F1.EmployeeID = F2.EmployeeID;

**5. Implement horizontal fragmentation in distributed DBMS.**

Horizontal fragmentation in a distributed database refers to dividing a table's rows into separate fragments, where each fragment contains a subset of rows based on a defined condition or predicate. This technique allows distributing different subsets of data across multiple database nodes or servers.

To perform horizontal fragmentation, you can split the table into two fragments based on the value of the "Country" column. Let's assume we have two database nodes, Node1 and Node2, and we want to fragment the data based on the country "USA" and "Canada".

Horizontal fragmentation allows for distributing data based on specific criteria, such as geographical location or other attributes. It enables parallel processing and data localization in a distributed environment.

**On Node1:**

CREATE TABLE Fragment1 AS

SELECT \*

FROM Customers

WHERE Country = 'USA';

**On Node 2:**

CREATE TABLE Fragment2 AS

SELECT \*

FROM Customers

WHERE Country = 'Canada';

SELECT \* FROM Fragment1;

SELECT \* FROM Fragment2;

**Related Practical Lab code:**

create table emptab(e\_id int,e\_name varchar(20),e\_sal number,e\_address varchar(20));

insert into emptab values(121,'A1',5000,'A1@gmail.com');

insert into emptab values(122,'A2',7000,'A2@gmail.com');

insert into emptab values(123,'A3',4000,'A3@gmail.com');

insert into emptab values(124,'A4',3500,'A4@gmail.com');

insert into emptab values(125,'A5',5000,'A5@gmail.com');

insert into emptab values(126,'A6',4500,'A6@gmail.com');

insert into emptab values(127,'A7',7500,'A7@gmail.com');

insert into emptab values(128,'A8',6500,'A8@gmail.com');

insert into emptab values(129,'A9',4000,'A9@gmail.com');

insert into emptab values(130,'A10',3500,'A10@gmail.com');

select \* from emptab order by e\_id;

create table emptab2 as(select \* from emptab where e\_sal<5000);

select \* from emptab2;

create table emptab3 as(select \* from emptab where e\_sal>5000);

select \* from emptab3;

**6. Implementation of database security using SQL commands like (Grant and Revoke).**

**7. PL/SQL Program to Add Two Numbers (All PL/SQL program: using function or procedure).**

**7.1-Using function:**

-- Add two numbers using a built-in function

SELECT 10 + 20 AS sum FROM dual;

Note: Run this on SQL \* plus command line

-- Create a function to add two numbers

CREATE OR REPLACE FUNCTION add\_numbers(a NUMBER, b NUMBER)

RETURN NUMBER

IS

result NUMBER;

BEGIN

result := a + b;

RETURN result;

END;

/

-- Call the function to add two numbers and retrieve the result

VAR sum\_result NUMBER;

EXEC :sum\_result := add\_numbers(10, 20);

-- Display the result

SELECT 'The sum is: ' || :sum\_result AS result FROM dual;

**7.2 using procedure:**

-- Create a procedure to add two numbers

CREATE OR REPLACE PROCEDURE add\_num(

a NUMBER,

b NUMBER,

sum OUT NUMBER

)

IS

BEGIN

sum := a + b;

END;

/

-- Set the output format

SET SERVEROUTPUT ON;

-- Declare variables to hold input values and the result

VARIABLE num1 NUMBER;

VARIABLE num2 NUMBER;

VARIABLE result NUMBER;

-- Assign input values

EXEC :num1 := 10;

EXEC :num2 := 20;

-- Call the procedure to add two numbers

EXEC add\_num(:num1, :num2, :result);

-- Display the result

PRINT 'The sum is: ' || :result;

**8. PL/SQL Program for Prime Number.**

**8.1 using function** (run on 10 g)

-- Create a function to check if a number is prime

CREATE OR REPLACE FUNCTION is\_prime(n NUMBER)

RETURN BOOLEAN

IS

divisor NUMBER := 2;

BEGIN

WHILE divisor <= SQRT(n) LOOP

IF MOD(n, divisor) = 0 THEN

RETURN FALSE; -- Number is not prime

END IF;

divisor := divisor + 1;

END LOOP;

RETURN TRUE; -- Number is prime

END;

/

-- Declare a variable to hold the number

DECLARE

num NUMBER;

BEGIN

-- Iterate from 1 to 100

FOR num IN 1..100 LOOP

-- Check if the number is prime

IF is\_prime(num) THEN

DBMS\_OUTPUT.PUT\_LINE(num);

END IF;

END LOOP;

END;

/

**8.2 using procedure.**

**9.PL/SQL Program to Find Factorial of a Number**

**9.1- using function:**

CREATE OR REPLACE FUNCTION CalculateFactorial(

n IN NUMBER

) RETURN NUMBER

IS

factorial NUMBER;

BEGIN

IF n = 0 THEN

factorial := 1;

ELSE

factorial := n \* CalculateFactorial(n - 1);

END IF;

RETURN factorial;

END;

/

DECLARE

num NUMBER := 5;

fact NUMBER;

BEGIN

fact := CalculateFactorial(num);

DBMS\_OUTPUT.PUT\_LINE('The factorial of ' || num || ' is: ' || fact);

END;

/

**9.2 using procedure:**

**10.PL/SQL Program for Reverse of a Number**

**10.1- using function:**

CREATE OR REPLACE FUNCTION ReverseNumber(

num IN NUMBER

) RETURN NUMBER

IS

reversed\_num NUMBER := 0;

temp\_num NUMBER := num;

remainder NUMBER;

BEGIN

WHILE temp\_num <> 0 LOOP

remainder := MOD(temp\_num, 10);

reversed\_num := reversed\_num \* 10 + remainder;

temp\_num := FLOOR(temp\_num / 10);

END LOOP;

RETURN reversed\_num;

END;

/

DECLARE

number\_to\_reverse NUMBER := 12345;

reversed\_number NUMBER;

BEGIN

reversed\_number := ReverseNumber(number\_to\_reverse);

DBMS\_OUTPUT.PUT\_LINE('The reverse of ' || number\_to\_reverse || ' is: ' || reversed\_number);

END;

/

**10.2- using procedure:**

**11.PL/SQL Program for Fibonacci Series**

**11.1- using function:**

**11.2- using procedure:**

CREATE OR REPLACE PROCEDURE GenerateFibonacciSeries(

num\_terms IN NUMBER

)

IS

term1 NUMBER := 0;

term2 NUMBER := 1;

fibonacci NUMBER;

BEGIN

DBMS\_OUTPUT.PUT(term1 || ' ' || term2 || ' ');

FOR i IN 3..num\_terms LOOP

fibonacci := term1 + term2;

DBMS\_OUTPUT.PUT(fibonacci || ' ');

term1 := term2;

term2 := fibonacci;

END LOOP;

DBMS\_OUTPUT.NEW\_LINE;

END;

/

BEGIN

GenerateFibonacciSeries(10);

END;

/

**12.PL/SQL Program to Check Number is Odd or Even**

**12.1- Using Function:**

**12.2- using procedure:**

CREATE OR REPLACE PROCEDURE CheckOddEven(

num IN NUMBER

)

IS

BEGIN

IF MOD(num, 2) = 0 THEN

DBMS\_OUTPUT.PUT\_LINE(num || ' is an even number.');

ELSE

DBMS\_OUTPUT.PUT\_LINE(num || ' is an odd number.');

END IF;

END;

/

BEGIN

CheckOddEven(7);

CheckOddEven(10);

END;

/

**13.PL/SQL Program to Reverse a String**

**13.1- using function:**

CREATE OR REPLACE FUNCTION ReverseString(

input\_string IN VARCHAR2

) RETURN VARCHAR2

IS

reversed\_string VARCHAR2(4000);

BEGIN

FOR i IN REVERSE 1..LENGTH(input\_string) LOOP

reversed\_string := reversed\_string || SUBSTR(input\_string, i, 1);

END LOOP;

RETURN reversed\_string;

END;

/

DECLARE

original\_string VARCHAR2(100) := 'Hello, World!';

reversed\_string VARCHAR2(100);

BEGIN

reversed\_string := ReverseString(original\_string);

DBMS\_OUTPUT.PUT\_LINE('Original String: ' || original\_string);

DBMS\_OUTPUT.PUT\_LINE('Reversed String: ' || reversed\_string);

END;

/

**13.2- using procedure:**

**14.Pl/SQL Program for Palindrome Number**

**14.1- using function:**

CREATE OR REPLACE FUNCTION IsPalindromeNumber(

num IN NUMBER

) RETURN BOOLEAN

IS

original\_num VARCHAR2(100);

reversed\_num VARCHAR2(100);

BEGIN

original\_num := TO\_CHAR(num);

reversed\_num := '';

FOR i IN REVERSE 1..LENGTH(original\_num) LOOP

reversed\_num := reversed\_num || SUBSTR(original\_num, i, 1);

END LOOP;

IF original\_num = reversed\_num THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END;

/

DECLARE

number\_to\_check NUMBER := 12321;

is\_palindrome BOOLEAN;

BEGIN

is\_palindrome := IsPalindromeNumber(number\_to\_check);

IF is\_palindrome THEN

DBMS\_OUTPUT.PUT\_LINE(number\_to\_check || ' is a palindrome number.');

ELSE

DBMS\_OUTPUT.PUT\_LINE(number\_to\_check || ' is not a palindrome number.');

END IF;

END;

/

**14.2- using procedure:**

**15.PL/SQL Program to Swap two Numbers**

**15.1- using function:**

**15.2- using procedure:**

CREATE OR REPLACE PROCEDURE SwapNumbers(

num1 IN OUT NUMBER,

num2 IN OUT NUMBER

)

IS

temp NUMBER;

BEGIN

temp := num1;

num1 := num2;

num2 := temp;

DBMS\_OUTPUT.PUT\_LINE('After swapping - Number 1: ' || num1 || ', Number 2: ' || num2);

END;

/

DECLARE

number1 NUMBER := 10;

number2 NUMBER := 20;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Before swapping - Number 1: ' || number1 || ', Number 2: ' || number2);

SwapNumbers(number1, number2);

END;

/

**16.PL/SQL Program for Armstrong Number**

**16.1 – using function:**

declare

n number:=407;

s number:=0;

r number;

len number;

m number;

begin

m:=n;

len:=length(to\_char(n));

while n>0

loop

r:=mod(n,10);

s:=s+power(r,len);

n:=trunc(n/10);

end loop;

if m=s

then

dbms\_output.put\_line('armstrong number');

else

dbms\_output.put\_line('not armstrong number');

end if;

end;

/

**16.2- using procedure:**

**17.PL/SQL Program to Find Greatest of Three Numbers**

DECLARE

num1 NUMBER := 10;

num2 NUMBER := 20;

num3 NUMBER := 15;

greatest NUMBER;

BEGIN

IF num1 > num2 AND num1 > num3 THEN

greatest := num1;

ELSIF num2 > num1 AND num2 > num3 THEN

greatest := num2;

ELSE

greatest := num3;

END IF;

DBMS\_OUTPUT.PUT\_LINE('The greatest number is: ' || greatest);

END;

/

**17.1- using function:**

CREATE OR REPLACE FUNCTION greatest\_of\_3(num1 NUMBER, num2 NUMBER, num3 NUMBER)

RETURN NUMBER IS

max\_num NUMBER;

BEGIN

IF num1 >= num2 AND num1 >= num3 THEN

max\_num := num1;

ELSIF num2 >= num1 AND num2 >= num3 THEN

max\_num := num2;

ELSE

max\_num := num3;

END IF;

RETURN max\_num;

END;

--then run this by passing arguments.

SELECT greatest\_of\_3(10, 20, 30) AS max\_num FROM DUAL;

**17.2- using procedure:**

CREATE OR REPLACE PROCEDURE find\_greatest (

num1 IN NUMBER,

num2 IN NUMBER,

num3 IN NUMBER,

greatest\_num OUT NUMBER

) AS

BEGIN

IF num1 >= num2 AND num1 >= num3 THEN

greatest\_num := num1;

ELSIF num2 >= num1 AND num2 >= num3 THEN

greatest\_num := num2;

ELSE

greatest\_num := num3;

END IF;

END;

DECLARE

result NUMBER;

BEGIN

find\_greatest(10, 20, 30, result);

DBMS\_OUTPUT.PUT\_LINE('The greatest number is ' || result);

END;

**18.PL/SQL procedure to check whether the current day is a weekend or weekday**

**18.1- using function:**

**18.2- using procedure:**

CREATE OR REPLACE PROCEDURE CheckWeekendOrWeekday

IS

current\_day VARCHAR2(10);

is\_weekend BOOLEAN := FALSE;

BEGIN

current\_day := UPPER(TO\_CHAR(SYSDATE, 'DAY'));

IF current\_day = 'SATURDAY' OR current\_day = 'SUNDAY' THEN

is\_weekend := TRUE;

END IF;

IF is\_weekend THEN

DBMS\_OUTPUT.PUT\_LINE('Today is a weekend.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Today is a weekday.');

END IF;

END;

/

BEGIN

CheckWeekendOrWeekday;

END;

/

**MONGODB**

**Q1-**

Database name: movie

Collection1: film

{title,rating,director,genre,release\_year,}

collection 2: actor

{id,name,title,age,address,}

Insert at least 10 documents inside film and actor.

**Q2-**

Database name: company

Collection1: employee(at least 5 documents)

{id,name,salary,designation,address}

Collection2: transaction(at least 10 documents)

{id,remark,made\_by,date,payment type}

**Points to be remember**

db.collectionname.insertMany([{key:value},{},{}])

db.collectionname.drop()

db.collectionname.find()

Solution 1🡺

1-Create databse.

use movie

2-create collections

db.createCollection("film")

db.createCollection("actor")

show collections

3-

db.film.insertMany([

{title:"dog",rating:4.5,director:"suraj",genre:"comedy",release\_year:2014},

{title:"war",rating:4.3,director:"sajid",genre:"action",release\_year:2015},

.

.

.

])