**FINAL BATTLE**

1)creating tables deposit , branch, customers, borrow and basic operations

create database bankp;

use bankp;

create table deposit (actno varchar(5),cname varchar(18),

bname varchar(18), amount\_no int , adate date);

create table branch (bname varchar(18),city varchar(18));

create table customers(cname varchar(19),city varchar(18));

create table borrow (loanno varchar(5),cname varchar(18),bname varchar(18),

amount\_no int );

INSERT INTO deposit (actno, cname, bname, amount\_no, adate)

VALUES

('001', 'Jay', 'Main Branch', 5000, '2023-11-23'),

('002', 'Janhavi', 'Downtown Branch', 7500, '2023-11-24'),

('003', 'Bhairvi', 'East Branch', 10000, '2023-11-25');

INSERT INTO branch (bname, city)

VALUES

('Main Branch', 'Pune'),

('Downtown Branch', 'Mumbai'),

('East Branch', 'Banglore');

INSERT INTO customers (cname, city)

VALUES

('Jay', 'Pune'),

('Janhavi', 'Mumbai'),

('Bhairavi', 'Banglore');

INSERT INTO borrow (loanno, cname, bname, amount\_no)

VALUES

('101', 'Jay', 'Main Branch', 2000),

('102', 'Janhavi', 'Downtown Branch', 3000),

('103', 'Bhairavi', 'East Branch', 4000);

describe deposit;

describe branch;

select \* from deposit;

select \* from borrow;

select actno , amount\_no from deposit;

2) & 3)create tables job,employee ,deposit,borrow and perform retrieving ,detail of acc no opened between dates,job with min salary

create database occupation;

use occupation;

CREATE TABLE Job (

job\_id INT PRIMARY KEY,

job\_title VARCHAR(50),

min\_sal INT,

max\_sal INT

);

CREATE TABLE Employee (

emp\_no INT PRIMARY KEY,

emp\_name VARCHAR(50),

emp\_sal INT,

emp\_comm INT,

dept\_no INT

);

CREATE TABLE deposit (

a\_no INT PRIMARY KEY,

cname VARCHAR(50),

bname VARCHAR(50),

amount INT,

a\_date DATE

);

CREATE TABLE borrow (

loanno INT PRIMARY KEY,

cname VARCHAR(50),

bname VARCHAR(50),

amount INT

);

INSERT INTO Job (job\_id, job\_title, min\_sal, max\_sal)

VALUES

(1, 'Manager', 50000, 80000),

(2, 'Developer', 30000, 60000),

(3, 'Analyst', 25000, 50000);

INSERT INTO Employee (emp\_no, emp\_name, emp\_sal, emp\_comm, dept\_no)

VALUES

(101, 'John ', 60000, 2000, 1),

(102, 'Lex', 45000, NULL, 2),

(103, 'Priya', 55000, 1500, 1);

INSERT INTO Employee (emp\_no, emp\_name, emp\_sal, emp\_comm, dept\_no)

VALUES

(104, 'Jaival ', 60500, 2200, 20);

-- Insert sample data into the 'deposit' table

INSERT INTO deposit (a\_no, cname, bname, amount, a\_date)

VALUES

(1001, 'Meave', 'Main Branch', 5000, '2023-11-23'),

(1002, 'Travis', 'Downtown Branch', 7500, '2023-11-24'),

(1003, 'Barry', 'East Branch', 10000, '2023-11-25');

INSERT INTO deposit (a\_no, cname, bname, amount, a\_date)

VALUES

(1004, 'Maya', 'Main Branch', 5300, '2006-03-23'),

(1005, 'Tracy', 'Downtown Branch', 7200, '2006-05-24');

-- Insert sample data into the 'borrow' table

INSERT INTO borrow (loanno, cname, bname, amount)

VALUES

(201, 'Louis', 'Main Branch', 2000),

(202, 'Harry', 'Downtown Branch', 3000),

(203, 'Shawn', 'East Branch', 4000);

INSERT INTO borrow (loanno, cname, bname, amount)

VALUES

(204, 'Louisa', 'Karolbagh Branch', 2100),

(205, 'Harris', 'Karolbagh Branch', 3050);

select \*from Employee;

select \* from Job;

select \* from deposit;

select a\_no,amount

from deposit

where a\_date between '2006-01-01' and '2006-07-25';

select job\_title

from job

where min\_sal>4000;

select emp\_name AS en ,emp\_sal

from Employee

where dept\_no=20;

select sum(amount) AS total\_deposit

from deposit;

select sum(amount) AS total\_loan

from borrow

where bname="karolbagh Branch";

select max(amount) as max\_loan

from borrow

where bname= 'karolbagh Branch';

select count(distinct cname)as total\_cust

from deposit;

**4) To implement Single-row functions. (i) Create tables according to the need. (ii) Insert the data for all tables. (1) Write a query to display the current date. Label the column Date. (2) For each employee, display the employee number, job, salary, and salary increased by 15% and expressed as a whole number. Label the column New Salary (3) Modify your query to add a column that subtracts the old salary from the new salary. Label the column Increase**

-- Create Employee table

use singlerow;

CREATE TABLE Employee (

emp\_no INT PRIMARY KEY,

emp\_name VARCHAR(50),

job\_title VARCHAR(50),

emp\_sal INT

);

-- Insert sample data into the Employee table

INSERT INTO Employee (emp\_no, emp\_name, job\_title, emp\_sal)

VALUES

(101, 'Jaya', 'Manager', 60000),

(102, 'Janhavi', 'Developer', 45000),

(103, 'Biswas', 'Analyst', 55000);

-- Alter the Employee table to add a 'Date' column

ALTER TABLE Employee

ADD COLUMN Date DATE;

-- Update the 'Date' column with the current date

UPDATE Employee

SET Date = CURRENT\_DATE;

-- Display the modified Employee table

SELECT \* FROM Employee;

-- Display employee number, job, salary, new salary increased by 15%, and the increase with the current date

SELECT

emp\_no,

job\_title,

emp\_sal AS Salary,

ROUND(emp\_sal \* 1.15) AS "New Salary",

ROUND(emp\_sal \* 0.15) AS Increase,

Date

FROM Employee;

5) Explain and implement joins and use suitable tables.

-- Create the "shop" database

CREATE DATABASE IF NOT EXISTS shop;

-- Switch to the "shop" database

USE shop;

-- Create the "customers" table

CREATE TABLE IF NOT EXISTS customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(255),

Email VARCHAR(255),

Phone VARCHAR(15)

);

-- Create the "orders" table

CREATE TABLE IF NOT EXISTS orders (

OrderID INT PRIMARY KEY,

CustomerID INT,

OrderDate DATE,

TotalAmount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES customers(CustomerID)

);

-- Insert data into the "customers" table

INSERT INTO customers (CustomerID, CustomerName, Email, Phone)

VALUES

(1, 'John Doe', 'john.doe@email.com', '123-456-7890'),

(2, 'Jane Smith', 'jane.smith@email.com', '987-654-3210'),

(3, 'Bob Johnson', 'bob.johnson@email.com', '456-789-0123');

-- Insert data into the "orders" table

INSERT INTO orders (OrderID, CustomerID, OrderDate, TotalAmount)

VALUES

(101, 1, '2023-11-24', 150.50),

(102, 2, '2023-11-25', 75.20),

(103, 1, '2023-11-26', 200.00),

(104, 3, '2023-11-27', 120.75);

SELECT \*

FROM customers

CROSS JOIN orders;

SELECT \*

FROM customers

FULL JOIN orders ON customers.CustomerID = orders.CustomerID;

SELECT \*

FROM customers

RIGHT JOIN orders ON customers.CustomerID = orders.CustomerID;

SELECT \*

FROM customers

LEFT JOIN orders ON customers.CustomerID = orders.CustomerID;

SELECT \*

FROM customers

LEFT JOIN orders ON customers.CustomerID = orders.CustomerID;

SELECT \*

FROM customers

INNER JOIN orders ON customers.CustomerID = orders.CustomerID;

**6) Apply the concept of Aggregating Data using Group functions use suitable tables.**

-- Create tables (assuming these tables are already created)

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

emp\_salary DECIMAL(10, 2),

dept\_id INT

);

CREATE TABLE Departments (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(50)

);

-- Insert data

INSERT INTO Employees VALUES (1, 'John Doe', 50000.00, 1);

INSERT INTO Employees VALUES (2, 'Jane Smith', 60000.00, 2);

INSERT INTO Employees VALUES (3, 'Bob Johnson', 55000.00, 1);

INSERT INTO Employees VALUES (4, 'Alice White', 70000.00, 2);

INSERT INTO Employees VALUES (5, 'Charlie Brown', 65000.00, 1);

INSERT INTO Departments VALUES (1, 'HR');

INSERT INTO Departments VALUES (2, 'IT');

SELECT dept\_name, COUNT(emp\_id) AS employee\_count

FROM Employees

JOIN Departments ON Employees.dept\_id = Departments.dept\_id

GROUP BY dept\_name;

SELECT dept\_name, SUM(emp\_salary) AS total\_salary

FROM Employees

JOIN Departments ON Employees.dept\_id = Departments.dept\_id

GROUP BY dept\_name;

SELECT dept\_name, AVG(emp\_salary) AS avg\_salary

FROM Employees

JOIN Departments ON Employees.dept\_id = Departments.dept\_id

GROUP BY dept\_name;

SELECT dept\_name, MIN(emp\_salary) AS min\_salary, MAX(emp\_salary) AS max\_salary

FROM Employees

JOIN Departments ON Employees.dept\_id = Departments.dept\_id

GROUP BY dept\_name;

**07 solve and implement queries using the concept of sub query**

-- (Assuming these tables are already created)

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

emp\_salary DECIMAL(10, 2),

dept\_id INT

);

CREATE TABLE Departments (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(50)

);

-- Insert data

INSERT INTO Employees VALUES (1, 'John Doe', 50000.00, 1);

INSERT INTO Employees VALUES (2, 'Jane Smith', 60000.00, 2);

INSERT INTO Employees VALUES (3, 'Bob Johnson', 55000.00, 1);

INSERT INTO Employees VALUES (4, 'Alice White', 70000.00, 2);

INSERT INTO Employees VALUES (5, 'Charlie Brown', 65000.00, 1);

INSERT INTO Departments VALUES (1, 'HR');

INSERT INTO Departments VALUES (2, 'IT');

-- Show the average salary for each department along with the overall average salary

SELECT dept\_name,

AVG(emp\_salary) AS avg\_salary,

(SELECT AVG(emp\_salary) FROM Employees) AS overall\_avg\_salary

FROM Employees

JOIN Departments ON Employees.dept\_id = Departments.dept\_id

GROUP BY dept\_name;

-- Show employees with a salary greater than the average salary of their department

SELECT emp\_name, emp\_salary, dept\_name

FROM Employees

JOIN Departments ON Employees.dept\_id = Departments.dept\_id

WHERE emp\_salary > (SELECT AVG(emp\_salary) FROM Employees WHERE Employees.dept\_id = Departments.dept\_id);

-- Show the total salary for each department using a subquery in the FROM clause

SELECT dept\_name, total\_salary

FROM (SELECT dept\_id, SUM(emp\_salary) AS total\_salary FROM Employees GROUP BY dept\_id) AS Subquery

JOIN Departments ON Subquery.dept\_id = Departments.dept\_id;

-- Show departments with an average salary greater than a specified value

SELECT dept\_name, AVG(emp\_salary) AS avg\_salary

FROM Employees

JOIN Departments ON Employees.dept\_id = Departments.dept\_id

GROUP BY dept\_name

HAVING AVG(emp\_salary) > (SELECT AVG(emp\_salary) FROM Employees);

**8) Use Manipulating Data, consider and create tables as per requirements. (1) Give 10% interest to all depositors. (2) Give 10% interest to all depositors having branch pune. (3) Give 10% interest to all depositors living in nagpur and having branch city as ramtek**

create database deposit;

use deposit;

CREATE TABLE depositors (

depositor\_id INT PRIMARY KEY,

name VARCHAR(50),

branch VARCHAR(50),

city VARCHAR(50),

balance DECIMAL(10, 2)

);

CREATE TABLE branches (

branch\_name VARCHAR(50) PRIMARY KEY,

city VARCHAR(50)

);

-- Inserting data into the depositors table

INSERT INTO depositors (depositor\_id, name, branch, city, balance) VALUES

(1, 'John Doe', 'Pune', 'Nagpur', 1000.00),

(2, 'Jane Doe', 'Mumbai', 'Mumbai', 2000.00),

(3, 'Bob Smith', 'Pune', 'Nagpur', 1500.00),

(4, 'Alice Johnson', 'Nagpur', 'Ramtek', 3000.00),

(5, 'Charlie Brown', 'Pune', 'Nagpur', 1200.00);

-- Inserting data into the branches table

INSERT INTO branches (branch\_name, city) VALUES

('Pune', 'Pune'),

('Mumbai', 'Mumbai'),

('Nagpur', 'Nagpur'),

('Ramtek', 'Ramtek');

UPDATE depositors

SET balance = balance \* 1.10;

UPDATE depositors

SET balance = balance \* 1.10

WHERE branch = 'Pune';

UPDATE depositors

SET balance = balance \* 1.10

WHERE city = 'Nagpur' AND branch IN (SELECT branch\_name FROM branches WHERE city = 'Ramtek');

**9) 1. Write a PL/SQL Block that will get the salary of employee with employee number ‘105’ and display it on the Screen**

create database sample;

use sample ;

create table employee (

emp\_id int primary key,

emp\_name varchar(50),

salary int );

insert into employee(emp\_id,emp\_name,salary)values

(101, 'John Doe', 50000),

(102, 'Jane smith', 60000),

(103, 'Johni Doeli', 99000),

(105, 'lie right', 59000);

call getsal();

**procedure**

CREATE DEFINER=`root`@`localhost` PROCEDURE `getsal`()

begin

declare v\_salary int;

select salary into v\_salary

from employee

where emp\_id = 105 ;

select 'employee 105 salary :',v\_salary;

end

2. Write a PL/SQL block that prints 1 to 5 numbers Using WHILE Loop Statement

create database loops;

use loops;

call disp();

procedure

CREATE DEFINER=`root`@`localhost` PROCEDURE `disp`()

begin

declare counter int default 1;

while counter <=5 do

select counter as number;

set counter =counter +1;

end while ;

end

Write a PL/SQL block that implements Function

create database funct;

use funct;

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(50),

salary DECIMAL(10,2),

department\_id INT

);

INSERT INTO employees VALUES

(1, 'John Doe', 50000.00, 10),

(2, 'Jane Smith', 60000.00, 20),

(3, 'Bob Johnson', 55000.00, 10),

(4, 'Alice Williams', 70000.00, 10),

(5, 'Charlie Brown', 65000.00, 20),

(6, 'Eva Davis', 75000.00, 30);

SELECT avg\_sal(10) AS avg\_salary;

Function

CREATE DEFINER=`root`@`localhost` FUNCTION `avg\_sal`(dept\_id int) RETURNS decimal(10,2)

DETERMINISTIC

begin

declare total\_salary decimal(10,2);

declare total\_emp int;

declare avg\_sal decimal(10,2);

select sum(salary),count(\*)into total\_salary,total\_emp

from employees

where department\_id=dept\_id;

if total\_emp > 0 then

set avg\_sal=total\_salary/total\_emp;

else set avg\_sal =0;

end if;

return avg\_sal;

end

TRIGGERS INSERT ,UPDATE ,DELETE AFTER

create database funct;

use funct;

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(50),

salary DECIMAL(10,2),

department\_id INT

);

INSERT INTO employees VALUES

(1, 'John Doe', 50000.00, 10),

(2, 'Jane Smith', 60000.00, 20),

(3, 'Bob Johnson', 55000.00, 10),

(4, 'Alice Williams', 70000.00, 10),

(5, 'Charlie Brown', 65000.00, 20),

(6, 'Eva Davis', 75000.00, 30);

CREATE TABLE employee\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

action VARCHAR(50),

log\_time TIMESTAMP

);

select \* from employee\_log;

INSERT INTO employees VALUES

(7, 'Jays Doe', 50000.00, 20);

update employees

set employee\_name='sanika',salary=55000.00,department\_id=30

where employee\_id=7;

delete from employees

where employee\_id=6;

INSERT AFTER

CREATE DEFINER=`root`@`localhost` TRIGGER `employees\_AFTER\_INSERT` AFTER INSERT ON `employees` FOR EACH ROW BEGIN

INSERT INTO employee\_log (employee\_id, action, log\_time)

VALUES (NEW.employee\_id, 'Inserted', NOW());

END

After update

CREATE DEFINER=`root`@`localhost` TRIGGER `employees\_AFTER\_UPDATE` AFTER UPDATE ON `employees` FOR EACH ROW BEGIN

insert into employee\_log(employee\_id,action,log\_time)

values(NEW.employee\_id,'update',now());

END

Delete after

CREATE DEFINER=`root`@`localhost` TRIGGER `employees\_AFTER\_DELETE` AFTER DELETE ON `employees` FOR EACH ROW BEGIN

insert into employee\_log(employee\_id,action,log\_time)

values(OLD.employee\_id,'deleted',now());

END

**Triggers insert ,update ,delete before**

create database befores;

use befores;

create database funct;

use funct;

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(50),

salary DECIMAL(10,2),

department\_id INT

);

INSERT INTO employees VALUES

(1, 'John Doe', 50000.00, 10),

(2, 'Jane Smith', 60000.00, 20),

(3, 'Bob Johnson', 55000.00, 10),

(4, 'Alice Williams', 70000.00, 10),

(5, 'Charlie Brown', 65000.00, 20),

(6, 'Eva Davis', 75000.00, 30);

CREATE TABLE employee\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

action VARCHAR(50),

log\_time TIMESTAMP

);

INSERT INTO employees (employee\_id, employee\_name, department\_id)

VALUES (8, 'New Employee', 30);

SELECT \* FROM employees WHERE employee\_id = 8;

UPDATE employees

SET salary = 48000.00

WHERE employee\_id = 1;

SELECT \* FROM employees WHERE employee\_id = 1;

call deleteemp(1);

**insert before**

CREATE DEFINER=`root`@`localhost` TRIGGER `employees\_BEFORE\_INSERT` BEFORE INSERT ON `employees` FOR EACH ROW BEGIN

if new.salary is null then

set new.salary=50000.00;

end if;

END

**Update before**

CREATE DEFINER=`root`@`localhost` TRIGGER `employees\_BEFORE\_UPDATE` BEFORE UPDATE ON `employees` FOR EACH ROW BEGIN

IF NEW.salary < 50000.00 THEN

SET NEW.salary = 50000.00;

END IF;

END

**Delete Before**

CREATE DEFINER=`root`@`localhost` TRIGGER `employees\_BEFORE\_DELETE` BEFORE DELETE ON `employees` FOR EACH ROW BEGIN

insert into employee\_log(employee\_id,action,log\_time)

values(OLD.employee\_id,'about to delete',now());

END

Procedure

CREATE DEFINER=`root`@`localhost` PROCEDURE `deleteemp`(in employee\_id\_pa int)

BEGIN

delete from employees

where employee\_id=employee\_id\_pa;

END

23 Consider the table and solve the quries: Sailors( sid, sname, rating, age) Boats(bid, bname,color) Reserves(sid,bid,day) 1. Find the names of sailors who have reserved boat number 103 2. Find the names of sailors who have never reserved boat number 103 3. Find the names of sailors who have reserved a red boat 4. Find the colors of boats reserved by Lubbe

CREATE TABLE Sailors (

sid INT PRIMARY KEY,

sname VARCHAR(50),

rating INT,

age INT

);

CREATE TABLE Boats (

bid INT PRIMARY KEY,

bname VARCHAR(50),

color VARCHAR(20)

);

CREATE TABLE Reserves (

sid INT,

bid INT,

day DATE,

PRIMARY KEY (sid, bid),

FOREIGN KEY (sid) REFERENCES Sailors(sid),

FOREIGN KEY (bid) REFERENCES Boats(bid)

);

SELECT sname

FROM Sailors

WHERE sid IN (SELECT sid FROM Reserves WHERE bid = 103);

SELECT sname

FROM Sailors

WHERE sid NOT IN (SELECT sid FROM Reserves WHERE bid = 103);

SELECT DISTINCT s.sname

FROM Sailors s

JOIN Reserves r ON s.sid = r.sid

JOIN Boats b ON r.bid = b.bid

WHERE b.color = 'red';

SELECT DISTINCT b.color

FROM Boats b

JOIN Reserves r ON b.bid = r.bid

JOIN Sailors s ON r.sid = s.sid

WHERE s.sname = 'Lubber';

**24) Consider the table and solve the quries: Sailors( sid, sname, rating, age) Boats(bid, bname,color) Reserves(sid,bid,day) 1. Find the names of sailors who have reserved boat number 103 2. Find the names of sailors who have never reserved boat number 103 3. Find the names of sailors who have reserved a red boat 4. Find the colors of boats reserved by Lubber 24 Consider the table and solve the quries: Sailors( sid, sname, rating, age) Boats(bid, bname,color) Reserves(sid,bid,day) 1. Find the names of sailors who have reserved a red or a green boat 2. Find the names of sailors who have reserved both a red and a green boat 3. Find the names of sailors who have reserved at least two different boats 4. Find the sids of silors with age over 20 who have not reserved a red boa**

-- Assuming the tables are created as follows:

CREATE TABLE Sailors (

sid INT PRIMARY KEY,

sname VARCHAR(50),

rating INT,

age INT

);

CREATE TABLE Boats (

bid INT PRIMARY KEY,

bname VARCHAR(50),

color VARCHAR(20)

);

CREATE TABLE Reserves (

sid INT,

bid INT,

day DATE,

PRIMARY KEY (sid, bid),

FOREIGN KEY (sid) REFERENCES Sailors(sid),

FOREIGN KEY (bid) REFERENCES Boats(bid)

);

SELECT DISTINCT sname

FROM Sailors s

JOIN Reserves r ON s.sid = r.sid

JOIN Boats b ON r.bid = b.bid

WHERE b.color IN ('red', 'green');

SELECT sname

FROM Sailors

WHERE sid IN (

SELECT sid

FROM Reserves r

JOIN Boats b ON r.bid = b.bid

WHERE b.color = 'red'

INTERSECT

SELECT sid

FROM Reserves r

JOIN Boats b ON r.bid = b.bid

WHERE b.color = 'green'

);

SELECT sname

FROM Sailors

WHERE sid IN (

SELECT sid

FROM Reserves

GROUP BY sid

HAVING COUNT(DISTINCT bid) >= 2

);

SELECT sid

FROM Sailors

WHERE age > 20

AND sid NOT IN (

SELECT sid

FROM Reserves r

JOIN Boats b ON r.bid = b.bid

WHERE b.color = 'red'

);

**25 Flights(flno, from, to, distance, departs, arrives, price) Aircraft(aid, aname, cruisingrange) Certified(eid, aid) Employees(eid, ename, salary) 1. Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots). 2. Print the name and salary of every nonpilot whose salary is more than the average salary for pilots. 3. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles.**

-- Assuming the tables are created as follows:

CREATE TABLE Flights (

flno INT PRIMARY KEY,

flfrom VARCHAR(50),

flto VARCHAR(50),

distance INT,

departs TIMESTAMP,

arrives TIMESTAMP,

price DECIMAL(10, 2)

);

CREATE TABLE Aircraft (

aid INT PRIMARY KEY,

aname VARCHAR(50),

cruisingrange INT

);

CREATE TABLE Certified (

eid INT,

aid INT,

PRIMARY KEY (eid, aid),

FOREIGN KEY (eid) REFERENCES Employees(eid),

FOREIGN KEY (aid) REFERENCES Aircraft(aid)

);

CREATE TABLE Employees (

eid INT PRIMARY KEY,

ename VARCHAR(50),

salary DECIMAL(10, 2)

);

SELECT AVG(CASE WHEN e.eid IS NOT NULL THEN e.salary END) - AVG(e.salary) AS salary\_difference

FROM Employees e

LEFT JOIN Certified c ON e.eid = c.eid

WHERE c.eid IS NOT NULL;

SELECT e.ename, e.salary

FROM Employees e

LEFT JOIN Certified c ON e.eid = c.eid

WHERE c.eid IS NULL AND e.salary > (SELECT AVG(salary) FROM Employees WHERE eid IN (SELECT eid FROM Certified));

SELECT e.ename

FROM Employees e

JOIN Certified c ON e.eid = c.eid

JOIN Aircraft a ON c.aid = a.aid

WHERE a.cruisingrange > 1000

AND NOT EXISTS (

SELECT 1

FROM Certified c2

JOIN Aircraft a2 ON c2.aid = a2.aid

WHERE c2.eid = e.eid AND a2.cruisingrange <= 1000

);

**21 Execute DDL statements which demonstrate the use of views. Try to update the base table using its corresponding view. Also consider restrictions on updatable views and perform view creation from multiple tables**

-- Create tables

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

emp\_salary DECIMAL(10, 2),

dept\_id INT

);

CREATE TABLE Departments (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(50)

);

-- Insert data

INSERT INTO Employees VALUES (1, 'John Doe', 50000.00, 1);

INSERT INTO Employees VALUES (2, 'Jane Smith', 60000.00, 2);

INSERT INTO Employees VALUES (3, 'Bob Johnson', 55000.00, 1);

INSERT INTO Departments VALUES (1, 'HR');

INSERT INTO Departments VALUES (2, 'IT');

-- Create a view from a single table

CREATE VIEW HR\_Employees AS

SELECT emp\_id, emp\_name, emp\_salary

FROM Employees

WHERE dept\_id = 1;

-- Update the base table using the view

UPDATE HR\_Employees

SET emp\_salary = emp\_salary \* 1.10

WHERE emp\_id = 1;

-- Attempt to update the base table using the view (may fail if the view is not updatable)

-- Uncomment the following line to see the potential error:

-- UPDATE HR\_Employees SET emp\_salary = emp\_salary \* 1.10 WHERE emp\_id = 2;

-- Create a view from multiple tables

CREATE VIEW EmployeeDetails AS

SELECT e.emp\_id, e.emp\_name, e.emp\_salary, d.dept\_name

FROM Employees e

JOIN Departments d ON e.dept\_id = d.dept\_id;

-- Query the view

SELECT \* FROM EmployeeDetails;

**13)Implement nested sub queries. Perform a test for set membership (in, not in), set comparison (=some,**

-- Create tables

CREATE TABLE Students (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(50),

age INT

);

CREATE TABLE Courses (

course\_id INT PRIMARY KEY,

course\_name VARCHAR(50),

instructor VARCHAR(50)

);

-- Insert data

INSERT INTO Students VALUES (1, 'John Doe', 20);

INSERT INTO Students VALUES (2, 'Jane Smith', 22);

INSERT INTO Students VALUES (3, 'Bob Johnson', 21);

INSERT INTO Courses VALUES (101, 'Mathematics', 'Prof. Brown');

INSERT INTO Courses VALUES (102, 'Physics', 'Prof. White');

INSERT INTO Courses VALUES (103, 'History', 'Prof. Davis');

-- Set membership (IN, NOT IN)

-- Find students who are enrolled in Mathematics

SELECT student\_name

FROM Students

WHERE student\_id IN (SELECT student\_id FROM Enrollments WHERE course\_id = 101);

-- Find students who are not enrolled in Physics

SELECT student\_name

FROM Students

WHERE student\_id NOT IN (SELECT student\_id FROM Enrollments WHERE course\_id = 102);

-- Set comparison (< SOME, >= SOME, < ALL, etc.)

-- Find students who are older than at least one student in the Physics class

SELECT student\_name

FROM Students

WHERE age > SOME (SELECT age FROM Students WHERE student\_id IN (SELECT student\_id FROM Enrollments WHERE course\_id = 102));

-- Set cardinality (UNIQUE, NOT UNIQUE)

-- Find courses with unique instructors

SELECT course\_name

FROM Courses

WHERE instructor IN (SELECT DISTINCT instructor FROM Courses);

-- Find courses with non-unique instructors

SELECT course\_name

FROM Courses

WHERE instructor NOT IN (SELECT DISTINCT instructor FROM Courses);

**10 ) Write a PL/SQL block that implements Implicit Cursor. 2 Write a PL/SQL block that implements Explicit Cursor.**

**Implicit cursor**

-- Create the database

CREATE DATABASE university;

-- Switch to the newly created database

USE university;

-- Create the 'employees' table

CREATE TABLE IF NOT EXISTS employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

emp\_salary DECIMAL(10, 2)

);

-- Insert data into 'employees' table

INSERT INTO employees (emp\_id, emp\_name, emp\_salary) VALUES

(1, 'John Doe', 50000.00),

(2, 'Jane Smith', 60000.00),

(3, 'Bob Johnson', 55000.00);

-- Create the 'departments' table

CREATE TABLE IF NOT EXISTS departments (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(50)

);

-- Insert data into 'departments' table

INSERT INTO departments (dept\_id, dept\_name) VALUES

(1, 'HR'),

(2, 'IT');

CALL GetEmployeeInfo();

Procedure

CREATE DEFINER=root@localhost PROCEDURE GetEmployeeInfo()

BEGIN

DECLARE v\_emp\_name VARCHAR(255);

DECLARE v\_emp\_salary DECIMAL(10, 2);

-- Implicit Cursor example

SELECT emp\_name, emp\_salary

INTO v\_emp\_name, v\_emp\_salary

FROM employees

WHERE emp\_id = 1;

-- Display the retrieved data

SELECT 'Employee Name: ', v\_emp\_name, 'Employee Salary: ', v\_emp\_salary;

END

**Explict cursor procedure**

CREATE DEFINER=root@localhost PROCEDURE DisplayDepartments()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_dept\_id INT;

DECLARE v\_dept\_name VARCHAR(255);

-- Explicit Cursor example

DECLARE dept\_cursor CURSOR FOR

SELECT dept\_id, dept\_name

FROM departments;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN dept\_cursor;

dept\_loop: LOOP

FETCH dept\_cursor INTO v\_dept\_id, v\_dept\_name;

IF done THEN

LEAVE dept\_loop;

END IF;

-- Display the retrieved data

SELECT 'Department ID: ', v\_dept\_id, 'Department Name: ', v\_dept\_name;

END LOOP;

CLOSE dept\_cursor;

END

**16) Write a PL/SQL block that implements BEFORE DELETE TRIGGER Write a trigger to check the salary is not Zero or Negative. Write a Trigger that check the employee resides in a city 'Pune'**

create database emp\_mgmt;

use emp\_mgmt;

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

salary DECIMAL(10,2),

city VARCHAR(50)

);

-- Sample data

INSERT INTO employees VALUES (1, 50000, 'Pune'),

(2, 60000, 'Mumbai'),

(3, 70000, 'Delhi');

DELETE FROM employees WHERE employee\_id = 1; -- This should raise an error due to the city condition

DELETE FROM employees WHERE employee\_id = 2; -- This should raise an error due to the salary condition

DELETE FROM employees WHERE employee\_id = 3; -- This should succeed

Procedure

CREATE DEFINER=`root`@`localhost` PROCEDURE `delete\_emp`(IN mp\_id INT)

BEGIN

DECLARE v\_salary DECIMAL(10,2);

DECLARE v\_city VARCHAR(50);

-- Retrieving values of salary and city for the employee being deleted

SELECT salary, city INTO v\_salary, v\_city

FROM employees

WHERE employee\_id = emp\_id;

-- Checking salary condition

IF v\_salary <= 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot delete employee with zero or negative salary.';

END IF;

-- Checking city condition

IF v\_city = 'Pune' THEN

SIGNAL SQLSTATE '45001'

SET MESSAGE\_TEXT = 'Cannot delete employee residing in Pune.';

END IF;

DELETE FROM employees WHERE employee\_id = emp\_id;

END

Trigger

CREATE DEFINER=`root`@`localhost` TRIGGER `before\_delete\_check` BEFORE DELETE ON `employees` FOR EACH ROW BEGIN

DECLARE v\_salary DECIMAL(10,2);

DECLARE v\_city VARCHAR(50);

-- Retrieving values of salary and city for the employee being deleted

SELECT salary, city INTO v\_salary, v\_city

FROM employees

WHERE employee\_id = OLD.employee\_id;

-- Checking salary condition

IF v\_salary <= 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot delete employee with zero or negative salary.';

END IF;

-- Checking city condition

IF v\_city = 'Pune' THEN

SIGNAL SQLSTATE '45001'

SET MESSAGE\_TEXT = 'Cannot delete employee residing in Pune.';

END IF;

END