S1) Consider following Relation

**Account (Acc\_no, branch\_name,balance)**

**Branch(branch\_name,branch\_city,assets)**

**Customer(cust\_name,cust\_street,cust\_city)**

**Depositor(cust\_name,acc\_no)**

**Loan(loan\_no,branch\_name,amount)**

**Borrower(cust\_name,loan\_no)**

Create above tables with appropriate constraints like primary key, foreign key, not null etc.

1. Find the names of all branches in loan relation.
2. Find all loan numbers for loans made at ‘Wadia College’ Branch with loan amount > 12000.
3. Find all customers who have a loan from bank. Find their names,loan\_no and loan amount.
4. List all customers in alphabetical order who have loan from ‘Wadia College’ branch.
5. Display distinct cities of branch.

-- Create Account table

CREATE TABLE Account (

Acc\_no INT PRIMARY KEY, -- Account number as the primary key

branch\_name VARCHAR(100) NOT NULL, -- Branch name cannot be NULL

balance DECIMAL(10, 2) NOT NULL, -- Balance cannot be NULL

FOREIGN KEY (branch\_name) REFERENCES Branch(branch\_name) -- Foreign key to Branch table

);

-- Create Branch table

CREATE TABLE Branch (

branch\_name VARCHAR(100) PRIMARY KEY, -- Branch name as the primary key

branch\_city VARCHAR(100) NOT NULL, -- Branch city cannot be NULL

assets DECIMAL(15, 2) NOT NULL -- Assets cannot be NULL

);

-- Create Customer table

CREATE TABLE Customer (

cust\_name VARCHAR(100) PRIMARY KEY, -- Customer name as the primary key

cust\_street VARCHAR(100) NOT NULL, -- Customer street cannot be NULL

cust\_city VARCHAR(100) NOT NULL -- Customer city cannot be NULL

);

-- Create Depositor table (to establish relationship between Customer and Account)

CREATE TABLE Depositor (

cust\_name VARCHAR(100),

acc\_no INT,

PRIMARY KEY (cust\_name, acc\_no), -- Composite primary key

FOREIGN KEY (cust\_name) REFERENCES Customer(cust\_name), -- Foreign key to Customer

FOREIGN KEY (acc\_no) REFERENCES Account(Acc\_no) -- Foreign key to Account

);

-- Create Loan table

CREATE TABLE Loan (

loan\_no INT PRIMARY KEY, -- Loan number as the primary key

branch\_name VARCHAR(100) NOT NULL, -- Branch name cannot be NULL

amount DECIMAL(15, 2) NOT NULL, -- Loan amount cannot be NULL

FOREIGN KEY (branch\_name) REFERENCES Branch(branch\_name) -- Foreign key to Branch

);

-- Create Borrower table (to establish relationship between Customer and Loan)

CREATE TABLE Borrower (

cust\_name VARCHAR(100),

loan\_no INT,

PRIMARY KEY (cust\_name, loan\_no), -- Composite primary key

FOREIGN KEY (cust\_name) REFERENCES Customer(cust\_name), -- Foreign key to Customer

FOREIGN KEY (loan\_no) REFERENCES Loan(loan\_no) -- Foreign key to Loan

);

-- Inserting data into Branch table

INSERT INTO Branch (branch\_name, branch\_city, assets)

VALUES

('Wadia College', 'Pune', 5000000),

('Fergusson College', 'Pune', 3000000),

('University Road', 'Mumbai', 7000000);

-- Inserting data into Account table

INSERT INTO Account (Acc\_no, branch\_name, balance)

VALUES

(101, 'Wadia College', 15000.00),

(102, 'Fergusson College', 5000.00),

(103, 'University Road', 20000.00),

(104, 'Wadia College', 25000.00);

-- Inserting data into Customer table

INSERT INTO Customer (cust\_name, cust\_street, cust\_city)

VALUES

('John Doe', '123 Maple St', 'Pune'),

('Jane Smith', '456 Oak St', 'Pune'),

('Alan Turing', '789 Birch St', 'Mumbai');

-- Inserting data into Depositor table

INSERT INTO Depositor (cust\_name, acc\_no)

VALUES

('John Doe', 101),

('Jane Smith', 102),

('Alan Turing', 103),

('Jane Smith', 104);

-- Inserting data into Loan table

INSERT INTO Loan (loan\_no, branch\_name, amount)

VALUES

(201, 'Wadia College', 15000.00),

(202, 'Fergusson College', 8000.00),

(203, 'University Road', 30000.00),

(204, 'Wadia College', 25000.00);

-- Inserting data into Borrower table

INSERT INTO Borrower (cust\_name, loan\_no)

VALUES

('John Doe', 201),

('Alan Turing', 203),

('Jane Smith', 204);

SELECT DISTINCT branch\_name

FROM Loan;

SELECT loan\_no

FROM Loan

WHERE branch\_name = 'Wadia College'

AND amount > 12000;

SELECT C.cust\_name, L.loan\_no, L.amount

FROM Borrower B

JOIN Customer C ON B.cust\_name = C.cust\_name

JOIN Loan L ON B.loan\_no = L.loan\_no;

SELECT C.cust\_name

FROM Borrower B

JOIN Customer C ON B.cust\_name = C.cust\_name

JOIN Loan L ON B.loan\_no = L.loan\_no

WHERE L.branch\_name = 'Wadia College'

ORDER BY C.cust\_name;

SELECT DISTINCT branch\_city

FROM Branch;

S2) Consider following Relation

**Account (Acc\_no, branch\_name,balance)**

**Branch(branch\_name,branch\_city,assets)**

**Customer(cust\_name,cust\_street,cust\_city)**

**Depositor(cust\_name,acc\_no)**

**Loan(loan\_no,branch\_name,amount)**

**Borrower(cust\_name,loan\_no)**

Create above tables with appropriate constraints like primary key, foreign key, not null etc.

1. Find all customers who have both account and loan at bank.
2. Find all customers who have an account or loan or both at bank.
3. Find all customers who have account but no loan at the bank.
4. Find average account balance at ‘Wadia College’ branch.

Find no. of depositors at each branch

SELECT DISTINCT D.cust\_name

FROM Depositor D

JOIN Borrower B ON D.cust\_name = B.cust\_name;

SELECT DISTINCT cust\_name

FROM Depositor

UNION

SELECT DISTINCT cust\_name

FROM Borrower;

SELECT DISTINCT D.cust\_name

FROM Depositor D

LEFT JOIN Borrower B ON D.cust\_name = B.cust\_name

WHERE B.loan\_no IS NULL;

SELECT AVG(balance) AS avg\_balance

FROM Account

WHERE branch\_name = 'Wadia College';

SELECT A.branch\_name, COUNT(DISTINCT D.cust\_name) AS num\_depositors

FROM Account A

JOIN Depositor D ON A.Acc\_no = D.acc\_no

GROUP BY A.branch\_name;

S3) Consider following Relation

**Account (Acc\_no, branch\_name,balance)**

**Branch(branch\_name,branch\_city,assets)**

**Customer(cust\_name,cust\_street,cust\_city)**

**Depositor(cust\_name,acc\_no)**

**Loan(loan\_no,branch\_name,amount)**

**Borrower(cust\_name,loan\_no)**

Create above tables with appropriate constraints like primary key, foreign key, not null etc.

1. Find the branches where average account balance > 15000.
2. Find number of tuples in customer relation.
3. Calculate total loan amount given by bank.
4. Delete all loans with loan amount between 1300 and 1500.
5. Find the average account balance at each branch
6. Find name of Customer and city where customer name starts with Letter P.

SELECT A.branch\_name

FROM Account A

GROUP BY A.branch\_name

HAVING AVG(A.balance) > 15000;

SELECT COUNT(\*) AS num\_customers

FROM Customer;

SELECT SUM(amount) AS total\_loan\_amount

FROM Loan;

DELETE FROM Loan

WHERE amount BETWEEN 1300 AND 1500;

SELECT branch\_name, AVG(balance) AS avg\_balance

FROM Account

GROUP BY branch\_name;

SELECT cust\_name, cust\_city

FROM Customer

WHERE cust\_name LIKE 'P%';

S7)

Consider following Relation

**Account (Acc\_no, branch\_name,balance)**

**Branch(branch\_name,branch\_city,assets)**

**Customer(cust\_name,cust\_street,cust\_city)**

**Depositor(cust\_name,acc\_no)**

**Loan(loan\_no,branch\_name,amount)**

**Borrower(cust\_name,loan\_no)** Execute the following query:

1. Create a View1 to display List all customers in alphabetical order who have loan from Pune\_Station branch.
2. Create View2 on branch table by selecting any two columns and perform insert update delete operations.
3. Create View3 on borrower and depositor table by selecting any one column from each table perform insert update delete operations.
4. Create Union of left and right joint for all customers who have an account or loan or both at bank
5. Create Simple and Unique index.
6. Display index Information.

7.

CREATE VIEW View1 AS

SELECT C.cust\_name

FROM Customer C

JOIN Borrower B ON C.cust\_name = B.cust\_name

JOIN Loan L ON B.loan\_no = L.loan\_no

WHERE L.branch\_name = 'Pune\_Station'

ORDER BY C.cust\_name;

-- Create View2

CREATE VIEW View2 AS

SELECT branch\_name, branch\_city

FROM Branch;

-- Insert a new branch into the Branch table through the View

INSERT INTO View2 (branch\_name, branch\_city)

VALUES ('New Branch', 'Chandigarh');

-- Update branch city for a specific branch

UPDATE View2

SET branch\_city = 'Mumbai'

WHERE branch\_name = 'New Branch';

-- Delete a branch from the Branch table through the View

DELETE FROM View2

WHERE branch\_name = 'New Branch';

-- Create View3

CREATE VIEW View3 AS

SELECT B.cust\_name, D.acc\_no

FROM Borrower B

JOIN Depositor D ON B.cust\_name = D.cust\_name;

-- Insert a new customer-loan account relation through the View

INSERT INTO View3 (cust\_name, acc\_no)

VALUES ('John Doe', 101);

-- Update the account number for a specific customer

UPDATE View3

SET acc\_no = 105

WHERE cust\_name = 'John Doe';

-- Delete a customer-loan record from the View

DELETE FROM View3

WHERE cust\_name = 'John Doe';

-- Create Union of LEFT JOIN and RIGHT JOIN

CREATE VIEW UnionView AS

SELECT D.cust\_name

FROM Depositor D

LEFT JOIN Borrower B ON D.cust\_name = B.cust\_name

UNION

SELECT B.cust\_name

FROM Borrower B

RIGHT JOIN Depositor D ON B.cust\_name = D.cust\_name;

-- Create a simple index on cust\_name column

CREATE INDEX idx\_cust\_name ON Customer(cust\_name);

-- Create a unique index on acc\_no column

CREATE UNIQUE INDEX idx\_unique\_acc\_no ON Account(Acc\_no);

-- Show index information for a table

SHOW INDEXES FROM Customer;

-- Query the INFORMATION\_SCHEMA to get detailed index info

SELECT \* FROM INFORMATION\_SCHEMA.STATISTICS

WHERE table\_name = 'Customer';

S4)

**SQL Queries:**

Create following tables with suitable constraints (primary key, foreign key, not null etc).

Insert record and solve the following queries:

**Create table Cust\_Master(Cust\_no, Cust\_name, Cust\_addr)**

**Create table Order(Order\_no, Cust\_no, Order\_date, Qty\_Ordered)**

**Create Product (Product\_no, Product\_name, Order\_no)**

1. List names of customers having 'A' as second letter in their name.
2. Display order from Customer no C1002, C1005, C1007 and C1008
3. List Clients who stay in either 'Banglore or 'Manglore'
4. Display name of customers& the product\_name they have purchase
5. Create view View1 consisting of Cust\_name, Product\_name.
6. Disply product\_name and quantity purchase by each customer
7. Perform different joint operation.

CREATE TABLE Cust\_Master ( Cust\_no VARCHAR(10) PRIMARY KEY, Cust\_name VARCHAR(100), Cust\_addr VARCHAR(255) );

CREATE TABLE Orders (

-> Order\_no VARCHAR(10) PRIMARY KEY,

-> Cust\_no VARCHAR(10),

-> Order\_date DATE,

-> Qty\_Ordered INT,

-> FOREIGN KEY (Cust\_no) REFERENCES Cust\_Master(Cust\_no)

-> );

create table Product(

-> Product\_no varchar(10) primary key,

-> Product\_name varchar(100),

-> Order\_no varchar(10),

-> foreign key (Order\_no) references Orders (Order\_no)

-> );

insert into Cust\_Master(Cust\_no, Cust\_name,Cust\_addr)

-> values

-> ('C1001','Aradhya','New York'),

-> ('C1002','Nitesh','Bangalore'),

-> ('C1003','Mahesh','Londan'),

-> ('C1004','Nitin','USA'),

-> ('C1005','Jeet','Punjab');

insert into Orders(Order\_no,Cust\_no,Order\_date,Qty\_Ordered)

-> values

-> ('1001','C1001','2024-10-01',5),

-> ('1002','C1002','2024-10-02',10),

-> ('1003','C1003','2024-10-03',15),

-> ('1004','C1004','2024-10-04',18),

-> ('1005','C1005','2024-10-05',25);

Query OK, 5 rows affected (0.01 sec)

insert into Product(Product\_no,Product\_name,Order\_no)values

-> ('P1001','Laptop','01001'),

-> ('P1002','Smartphones','01002'),

-> ('P1003','Headphones','01003'),

-> ('P1004','Monitor','01004'),

-> ('P1005','Tablets','01005');

SELECT Cust\_name

-> FROM Cust\_Master

-> WHERE LOCATE('A', Cust\_name, 2) = 2;

select \* from Orders

-> where Cust\_no in ('C1001','C1003','C1005');

select Cust\_name from Cust\_master

-> where Cust\_addr in('Bangalore','Londan');

select cm.Cust\_name,p.Product\_name from Cust\_Master cm

-> join Orders o on cm.Cust\_no = o.Cust\_no

-> join Product p on o.Order\_no = p.Order\_no;

create view View1 as

-> select cm.Cust\_name ,p.Product\_name

-> from Cust\_Master cm

-> join Orders o on cm.Cust\_no = o.Cust\_no

-> join Product p on o.Order\_no = p.Order\_no;

select cm.Cust\_name,p.Product\_name,sum(o.Qty\_Ordered) as Total\_Qty

-> from Cust\_Master cm

-> join Product p on o.Order\_no = p.Order\_no

-> join Orders o on cm.Cust\_no = o.Cust\_no

-> group by cm.Cust\_name,p.Product\_name;

SELECT cm.Cust\_name, p.Product\_name

-> FROM Cust\_Master cm

-> LEFT JOIN `Order` o ON cm.Cust\_no = o.Cust\_no

-> LEFT JOIN Product p ON o.Order\_no = p.Order\_no;

SELECT cm.Cust\_name, p.Product\_name

-> FROM Cust\_Master cm

-> RIGHT JOIN `Order` o ON cm.Cust\_no = o.Cust\_no

-> RIGHT JOIN Product p ON o.Order\_no = p.Order\_no;

S5-S6)

Consider following Relation

**Employee(emp\_id,employee\_name,street,city)**

**Works(employee\_name,company\_name,salary)**

**Company(company\_name,city)**

**Manages(employee\_name,manager\_name)**

Create above tables with appropriate constraints like primary key, foreign key, not null etc.

1. Find the names of all employees who work for ‘TCS’.
2. Find the names and company names of all employees sorted in ascending order of company name and descending order of employee names of that company.
3. Change the city of employee working with InfoSys to ‘Bangalore’
4. Find the names, street address, and cities of residence for all employees who work for 'TechM' and earn more than $10,000.
5. Add Column Asset to Company table.

-- Create Employee table

CREATE TABLE Employee (

emp\_id INT PRIMARY KEY, -- Employee ID as primary key

employee\_name VARCHAR(100) NOT NULL, -- Employee name cannot be null

street VARCHAR(255), -- Street address (nullable)

city VARCHAR(100) -- City (nullable)

);

-- Create Company table

CREATE TABLE Company (

company\_name VARCHAR(100) PRIMARY KEY, -- Company name as primary key

city VARCHAR(100), -- City of the company (nullable)

asset DECIMAL(15, 2) -- Asset column will be added in Step 5

);

-- Create Works table

CREATE TABLE Works (

employee\_name VARCHAR(100), -- Employee name as a foreign key reference

company\_name VARCHAR(100), -- Company name as a foreign key reference

salary DECIMAL(15, 2) NOT NULL, -- Salary should not be null

PRIMARY KEY (employee\_name, company\_name), -- Composite primary key

FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name), -- Foreign key from Employee

FOREIGN KEY (company\_name) REFERENCES Company(company\_name) -- Foreign key from Company

);

-- Create Manages table

CREATE TABLE Manages (

employee\_name VARCHAR(100), -- Employee name who manages

manager\_name VARCHAR(100), -- Manager's name

PRIMARY KEY (employee\_name, manager\_name), -- Composite primary key

FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name), -- Foreign key from Employee

FOREIGN KEY (manager\_name) REFERENCES Employee(employee\_name) -- Foreign key from Employee

);

-- Insert data into Employee table

INSERT INTO Employee (emp\_id, employee\_name, street, city) VALUES

(1, 'John Doe', '123 Main St', 'New York'),

(2, 'Jane Smith', '456 Oak Ave', 'Los Angeles'),

(3, 'Mike Johnson', '789 Pine Rd', 'San Francisco'),

(4, 'Lucy Brown', '101 Maple Dr', 'Chicago');

-- Insert data into Company table

INSERT INTO Company (company\_name, city, asset) VALUES

('TCS', 'Mumbai', 1000000.00),

('Infosys', 'Bangalore', 2000000.00),

('TechM', 'Hyderabad', 1500000.00);

-- Insert data into Works table

INSERT INTO Works (employee\_name, company\_name, salary) VALUES

('John Doe', 'TCS', 12000.00),

('Jane Smith', 'Infosys', 9500.00),

('Mike Johnson', 'TechM', 15000.00),

('Lucy Brown', 'TCS', 8500.00);

-- Insert data into Manages table

INSERT INTO Manages (employee\_name, manager\_name) VALUES

('John Doe', 'Mike Johnson'),

('Jane Smith', 'Lucy Brown');

SELECT employee\_name

FROM Works

WHERE company\_name = 'TCS';

SELECT employee\_name, company\_name

FROM Works

ORDER BY company\_name ASC, employee\_name DESC;

UPDATE Employee

SET city = 'Bangalore'

WHERE employee\_name IN (SELECT employee\_name FROM Works WHERE company\_name = 'Infosys');

SELECT e.employee\_name, e.street, e.city

FROM Employee e

JOIN Works w ON e.employee\_name = w.employee\_name

WHERE w.company\_name = 'TechM' AND w.salary > 10000;

ALTER TABLE Company

ADD COLUMN asset DECIMAL(15, 2);

-- Create Employee table

CREATE TABLE Employee (

emp\_id INT AUTO\_INCREMENT PRIMARY KEY, -- Employee ID as the primary key (auto-incremented)

employee\_name VARCHAR(100) NOT NULL, -- Employee name cannot be NULL

street VARCHAR(255), -- Street address (nullable)

city VARCHAR(100) -- City (nullable)

);

-- Create Company table

CREATE TABLE Company (

company\_name VARCHAR(100) PRIMARY KEY, -- Company name as the primary key

city VARCHAR(100), -- City of the company (nullable)

asset DECIMAL(15, 2) -- Asset column (nullable)

);

-- Create Works table

CREATE TABLE Works (

employee\_name VARCHAR(100), -- Employee name (foreign key)

company\_name VARCHAR(100), -- Company name (foreign key)

salary DECIMAL(15, 2) NOT NULL, -- Salary cannot be null

PRIMARY KEY (employee\_name, company\_name), -- Composite primary key

FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name), -- Foreign key from Employee

FOREIGN KEY (company\_name) REFERENCES Company(company\_name) -- Foreign key from Company

);

-- Create Management table (previously Manages)

CREATE TABLE Management (

employee\_name VARCHAR(100), -- Employee name (foreign key)

manager\_name VARCHAR(100), -- Manager name (foreign key)

PRIMARY KEY (employee\_name, manager\_name), -- Composite primary key

FOREIGN KEY (employee\_name) REFERENCES Employee(employee\_name), -- Foreign key from Employee

FOREIGN KEY (manager\_name) REFERENCES Employee(employee\_name) -- Foreign key from Employee

);

UPDATE Employee

SET city = 'Bangalore'

WHERE employee\_name IN (SELECT employee\_name FROM Works WHERE company\_name = 'Infosys');

SELECT W.employee\_name

FROM Works W

JOIN Company C ON W.company\_name = C.company\_name

WHERE W.salary > (SELECT AVG(salary) FROM Works WHERE company\_name = W.company\_name);

SELECT E.employee\_name, E.street, E.city

FROM Employee E

JOIN Works W ON E.employee\_name = W.employee\_name

WHERE W.company\_name = 'TechM' AND W.salary > 10000;

RENAME TABLE Manages TO Management;

-- Simple index on employee\_name

CREATE INDEX idx\_employee\_name ON Employee(employee\_name);

-- Unique index on employee\_name (to ensure uniqueness)

CREATE UNIQUE INDEX uniq\_employee\_name ON Employee(employee\_name);

SHOW INDEX FROM Employee;

S8)

Consider following Relation:

**Companies (comp\_id, name, cost, year) Orders (comp\_id, domain, quantity)** Execute the following query:

1. Find names, costs, domains and quantities for companies using inner join.
2. Find names, costs, domains and quantities for companies using left outer join.
3. Find names, costs, domains and quantities for companies using right outer join.
4. Find names, costs, domains and quantities for companies using Union operator.
5. Create View View1 by selecting both tables to show company name and quantities.
6. Create View View2 by selecting any two columns and perform insert update delete operations.
7. Display content of View1, View2.

-- Create Companies table

CREATE TABLE Companies (

comp\_id INT PRIMARY KEY, -- Company ID as primary key

name VARCHAR(100) NOT NULL, -- Company name

cost DECIMAL(15, 2), -- Cost of the company

year INT -- Year of establishment

);

-- Create Orders table

CREATE TABLE Orders (

comp\_id INT, -- Company ID (foreign key reference)

domain VARCHAR(100), -- Domain of the company

quantity INT, -- Quantity of orders

FOREIGN KEY (comp\_id) REFERENCES Companies(comp\_id) -- Foreign key reference to Companies table

);

-- Insert sample data into Companies table

INSERT INTO Companies (comp\_id, name, cost, year) VALUES

(1, 'CompanyA', 150000.00, 1995),

(2, 'CompanyB', 200000.00, 2000),

(3, 'CompanyC', 300000.00, 2010);

-- Insert sample data into Orders table

INSERT INTO Orders (comp\_id, domain, quantity) VALUES

(1, 'Electronics', 1000),

(2, 'Software', 2000),

(3, 'Healthcare', 1500);

SELECT c.name, c.cost, o.domain, o.quantity

FROM Companies c

INNER JOIN Orders o ON c.comp\_id = o.comp\_id;

SELECT c.name, c.cost, o.domain, o.quantity

FROM Companies c

LEFT OUTER JOIN Orders o ON c.comp\_id = o.comp\_id;

SELECT c.name, c.cost, o.domain, o.quantity

FROM Companies c

RIGHT OUTER JOIN Orders o ON c.comp\_id = o.comp\_id;

SELECT c.name, c.cost, o.domain, o.quantity

FROM Companies c

JOIN Orders o ON c.comp\_id = o.comp\_id

UNION

SELECT c.name, c.cost, NULL, NULL

FROM Companies c;

CREATE VIEW View1 AS

SELECT c.name, o.quantity

FROM Companies c

INNER JOIN Orders o ON c.comp\_id = o.comp\_id;

CREATE VIEW View2 AS

SELECT name, cost

FROM Companies;

-- Insert Operation

INSERT INTO View2 (name, cost) VALUES ('CompanyD', 400000.00);

-- Update Operation

UPDATE View2

SET cost = 250000.00

WHERE name = 'CompanyA';

-- Delete Operation

DELETE FROM View2 WHERE name = 'CompanyB';

SELECT \* FROM View1;

SELECT \* FROM View2;

S9) **SQL Queries**

Create following tables with suitable constraints. Insert data and solve the following queries:

**CUSTOMERS(CNo, Cname, Ccity, CMobile)**

**ITEMS(INo, Iname, Itype, Iprice, Icount)**

**PURCHASE(PNo, Pdate, Pquantity, Cno, INo)**

1. List all stationary items with price between 400/- to 1000/-
2. Change the mobile number of customer “Gopal”
3. Display the item with maximum price
4. Display all purchases sorted from the most recent to the oldest
5. Count the number of customers in every city
6. Display all purchased quantity of Customer Maya
7. Create view which shows Iname, Price and Count of all stationary items in descending order of price.

-- Create the CUSTOMERS table

CREATE TABLE CUSTOMERS (

CNo INT PRIMARY KEY, -- Customer Number (Primary Key)

Cname VARCHAR(50), -- Customer Name

Ccity VARCHAR(50), -- Customer City

CMobile VARCHAR(15) -- Customer Mobile Number

);

-- Create the ITEMS table

CREATE TABLE ITEMS (

INo INT PRIMARY KEY, -- Item Number (Primary Key)

Iname VARCHAR(100), -- Item Name

Itype VARCHAR(50), -- Item Type (e.g., Stationary, Electronics, etc.)

Iprice DECIMAL(10, 2), -- Item Price

Icount INT -- Available Stock Quantity

);

-- Create the PURCHASE table

CREATE TABLE PURCHASE (

PNo INT PRIMARY KEY, -- Purchase Number (Primary Key)

Pdate DATE, -- Purchase Date

Pquantity INT, -- Quantity Purchased

CNo INT, -- Customer Number (Foreign Key)

INo INT, -- Item Number (Foreign Key)

FOREIGN KEY (CNo) REFERENCES CUSTOMERS(CNo),

FOREIGN KEY (INo) REFERENCES ITEMS(INo)

);

-- Insert data into the CUSTOMERS table

INSERT INTO CUSTOMERS (CNo, Cname, Ccity, CMobile) VALUES

(1, 'Gopal', 'Delhi', '9876543210'),

(2, 'Maya', 'Mumbai', '9988776655'),

(3, 'Raj', 'Bangalore', '9123456789'),

(4, 'Sita', 'Delhi', '9776543210');

-- Insert data into the ITEMS table

INSERT INTO ITEMS (INo, Iname, Itype, Iprice, Icount) VALUES

(101, 'Notebook', 'Stationary', 450.00, 100),

(102, 'Pen', 'Stationary', 50.00, 500),

(103, 'Eraser', 'Stationary', 20.00, 200),

(104, 'Laptop', 'Electronics', 50000.00, 30),

(105, 'Desk Organizer', 'Stationary', 800.00, 50),

(106, 'Charger', 'Electronics', 1200.00, 100);

-- Insert data into the PURCHASE table

INSERT INTO PURCHASE (PNo, Pdate, Pquantity, CNo, INo) VALUES

(1, '2024-10-01', 2, 1, 101),

(2, '2024-10-02', 5, 2, 102),

(3, '2024-10-03', 1, 1, 105),

(4, '2024-10-04', 3, 3, 101),

(5, '2024-10-05', 10, 4, 106),

(6, '2024-10-06', 4, 2, 103);

SELECT Iname, Iprice

FROM ITEMS

WHERE Itype = 'Stationary'

AND Iprice BETWEEN 400 AND 1000;

UPDATE CUSTOMERS

SET CMobile = '9988776655'

WHERE Cname = 'Gopal';

SELECT Iname, Iprice

FROM ITEMS

WHERE Iprice = (SELECT MAX(Iprice) FROM ITEMS);

SELECT PNo, Pdate, Pquantity, CNo, INo

FROM PURCHASE

ORDER BY Pdate DESC;

SELECT Ccity, COUNT(\*) AS CustomerCount

FROM CUSTOMERS

GROUP BY Ccity;

SELECT P.PNo, P.Pdate, P.Pquantity, I.Iname

FROM PURCHASE P

JOIN ITEMS I ON P.INo = I.INo

JOIN CUSTOMERS C ON P.CNo = C.CNo

WHERE C.Cname = 'Maya';

CREATE VIEW StationaryItemsView AS

SELECT Iname, Iprice, Icount

FROM ITEMS

WHERE Itype = 'Stationary'

ORDER BY Iprice DESC;

**PL/SQL:-**

|  |  |
| --- | --- |
| P1)  **P1** | Write a **PL/SQL code** block to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area. |

SQL> show databases;

SP2-0158: unknown SHOW option "databases"

SQL> create table areas(

2 radius number(3,1),

3 area number(10,2)

4 );

Table created.

SQL> declare

2 radius\_value number;

3 area\_value number;

4 begin

5

6 for radius\_value in 5..9 loop

7 area\_value :=3.1416 \* (radius\_value \* radius\_value);

8

9 insert into areas(radius,area) values(radius\_value,area\_value);

10 end loop;

11 commit;

12

13 dbms\_output.put\_line('area for radii from 5 to 9 have been calculated and inserted.');

14 end;

15 /

PL/SQL procedure successfully completed.

SQL> select \* from areas;

P2)

Write an **Unnamed PL/SQL** of code for the following requirements: -

Schema:

Borrower (Rollin, Name, DateofIssue, NameofBook, Status)

Fine (Roll\_no,Date,Amt)

Accept roll\_no & name of book from user.

Check the number of days (from date of issue).

1. If days are between 15 to 30 then fine amounts will be Rs 5 per day.
2. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
3. After submitting the book, status will change from I to R.
4. If condition of fine is true, then details will be stored into fine table.

-- Create Borrower Table

CREATE TABLE Borrower (

Rollin NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DateofIssue DATE,

NameofBook VARCHAR2(100),

Status CHAR(1) CHECK (Status IN ('I', 'R')) -- 'I' for Issued, 'R' for Returned

);

-- Create Fine Table

CREATE TABLE Fine (

Roll\_no NUMBER,

FineDate DATE,

Amt NUMBER

);

-- Inserting sample data into Borrower table

INSERT INTO Borrower (Rollin, Name, DateofIssue, NameofBook, Status)

VALUES (101, 'John Doe', TO\_DATE('2024-10-01', 'YYYY-MM-DD'), 'PL/SQL Programming', 'I');

INSERT INTO Borrower (Rollin, Name, DateofIssue, NameofBook, Status)

VALUES (102, 'Alice Smith', TO\_DATE('2024-09-25', 'YYYY-MM-DD'), 'Advanced SQL', 'I');

DECLARE

v\_roll\_no NUMBER := &roll\_no; -- User input for roll number

v\_book\_name VARCHAR2(100) := '&book\_name'; -- User input for book name

v\_issue\_date DATE;

v\_status CHAR(1);

v\_fine NUMBER := 0;

v\_days NUMBER;

BEGIN

-- Retrieve the issue date and status for the given roll number and book

SELECT DateofIssue, Status

INTO v\_issue\_date, v\_status

FROM Borrower

WHERE Rollin = v\_roll\_no AND NameofBook = v\_book\_name;

-- Proceed only if the book is still issued ('I')

IF v\_status = 'I' THEN

v\_days := TRUNC(SYSDATE - v\_issue\_date); -- Calculate the number of days

-- Calculate fine based on days

IF v\_days BETWEEN 15 AND 30 THEN

v\_fine := v\_days \* 5; -- Rs. 5 per day

ELSIF v\_days > 30 THEN

v\_fine := 30 \* 5 + (v\_days - 30) \* 50; -- Rs. 5 for first 30 days, Rs. 50 thereafter

END IF;

-- Update status to 'R' (Returned) and record fine if applicable

UPDATE Borrower SET Status = 'R' WHERE Rollin = v\_roll\_no AND NameofBook = v\_book\_name;

IF v\_fine > 0 THEN

INSERT INTO Fine (Roll\_no, Date, Amt) VALUES (v\_roll\_no, SYSDATE, v\_fine);

END IF;

COMMIT; -- Commit changes

-- Output the fine

DBMS\_OUTPUT.PUT\_LINE('Book returned. Fine: Rs. ' || v\_fine);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Error: Book not issued or already returned.');

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No record found for the given Roll No. and Book Name.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

P3)

Write a **PL/SQL block** of code using Cursor that will merge the data available in the newly created table N\_Roll Call with the data available in the table O\_RollCall. If the data in the first table already exist in the second table, then that data should be skipped.

-- Create the N\_RollCall table

CREATE TABLE N\_RollCall (

Roll\_no NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Date\_of\_Call DATE

);

-- Create the O\_RollCall table

CREATE TABLE O\_RollCall (

Roll\_no NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Date\_of\_Call DATE

);

-- Insert sample data into N\_RollCall table

INSERT INTO N\_RollCall (Roll\_no, Name, Date\_of\_Call)

VALUES (1, 'John Doe', TO\_DATE('2024-11-01', 'YYYY-MM-DD'));

INSERT INTO N\_RollCall (Roll\_no, Name, Date\_of\_Call)

VALUES (2, 'Alice', TO\_DATE('2024-11-02', 'YYYY-MM-DD'));

-- Insert initial data into O\_RollCall table

INSERT INTO O\_RollCall (Roll\_no, Name, Date\_of\_Call)

VALUES (1, 'John Doe', TO\_DATE('2024-11-01', 'YYYY-MM-DD'));

DECLARE

CURSOR c\_new\_rollcall IS

SELECT \* FROM N\_RollCall;

-- Variable to hold each row from the cursor

v\_new\_row c\_new\_rollcall%ROWTYPE;

BEGIN

-- Open the cursor and loop through each row of the N\_RollCall table

FOR v\_new\_row IN c\_new\_rollcall LOOP

-- Check if the record already exists in O\_RollCall

IF NOT EXISTS (SELECT 1 FROM O\_RollCall WHERE some\_unique\_column = v\_new\_row.some\_unique\_column) THEN

-- If not exists, insert the row from N\_RollCall into O\_RollCall

INSERT INTO O\_RollCall (col1, col2, col3) -- Replace with actual column names

VALUES (v\_new\_row.col1, v\_new\_row.col2, v\_new\_row.col3); -- Replace with actual column names

END IF;

END LOOP;

-- Commit the transaction to apply the changes

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Merge operation completed successfully.');

EXCEPTION

WHEN OTHERS THEN

-- Handle any errors

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

P4)

Write a **PL/SQL block** for following requirements and handle the exceptions. Roll no. of students will be entered by the user. Attendance of roll no. entered by user will be checked in the Stud table. If attendance is less than 75% then display the message “Term not granted” and set the status in stud table as “Detained”. Otherwise display message “Term granted” and set the status in stud table as “Not Detained”. **Student (Roll, Name, Attendance, Status)**

CREATE TABLE Stud (

Roll NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Attendance NUMBER,

Status VARCHAR2(20)

);

INSERT INTO Stud (Roll, Name, Attendance, Status)

VALUES (1, 'John Doe', 80, 'Not Detained');

INSERT INTO Stud (Roll, Name, Attendance, Status)

VALUES (2, 'Jane Smith', 60, 'Not Detained');

INSERT INTO Stud (Roll, Name, Attendance, Status)

VALUES (3, 'Alice Johnson', 90, 'Not Detained');

INSERT INTO Stud (Roll, Name, Attendance, Status)

VALUES (4, 'Bob Brown', 72, 'Not Detained');

COMMIT;

DECLARE

-- Declare variables to store student details

v\_roll\_no NUMBER;

v\_attendance NUMBER;

v\_status VARCHAR2(20);

BEGIN

-- Prompt user for roll number (in SQL\*Plus, SQLcl, etc. this will take input)

DBMS\_OUTPUT.PUT\_LINE('Enter Roll No:');

-- For illustration, use a fixed value for roll number (adjust as needed)

-- In actual use, this value would be taken from user input in a real environment

v\_roll\_no := &roll\_no; -- Assume roll\_no is entered by the user via a substitution variable

-- Fetch attendance of the student from the Stud table

BEGIN

SELECT Attendance INTO v\_attendance

FROM Stud

WHERE Roll = v\_roll\_no;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No student found with the provided Roll No.');

RETURN;

END;

-- Determine the status based on attendance

IF v\_attendance < 75 THEN

v\_status := 'Detained';

DBMS\_OUTPUT.PUT\_LINE('Term not granted. Attendance is less than 75%.');

ELSE

v\_status := 'Not Detained';

DBMS\_OUTPUT.PUT\_LINE('Term granted. Attendance is 75% or more.');

END IF;

-- Update the status of the student in the Stud table

UPDATE Stud

SET Status = v\_status

WHERE Roll = v\_roll\_no;

-- Commit the transaction to save changes

COMMIT;

EXCEPTION

WHEN OTHERS THEN

-- Handle any unexpected errors

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

P5)

Write a **PL/SQL Block** to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, whenever such salary updates take place, a record for same is maintained in the increment\_salary table.

emp(emp\_no, salary) increment\_salary(emp\_no, salary)

-- Create emp table

CREATE TABLE emp (

emp\_no NUMBER PRIMARY KEY,

salary NUMBER

);

-- Create increment\_salary table to log salary increments

CREATE TABLE increment\_salary (

emp\_no NUMBER,

old\_salary NUMBER,

new\_salary NUMBER,

increment\_date DATE DEFAULT SYSDATE

);

-- Insert sample employee data into emp table

INSERT INTO emp (emp\_no, salary) VALUES (1, 3000);

INSERT INTO emp (emp\_no, salary) VALUES (2, 5000);

INSERT INTO emp (emp\_no, salary) VALUES (3, 4000);

INSERT INTO emp (emp\_no, salary) VALUES (4, 6000);

INSERT INTO emp (emp\_no, salary) VALUES (5, 4500);

COMMIT;

DECLARE

v\_avg\_salary NUMBER;

v\_old\_salary NUMBER;

v\_new\_salary NUMBER;

BEGIN

-- Calculate the average salary of the organization

SELECT AVG(salary) INTO v\_avg\_salary FROM emp;

-- Loop through employees whose salary is less than the average salary

FOR emp\_rec IN (SELECT emp\_no, salary FROM emp WHERE salary < v\_avg\_salary) LOOP

v\_old\_salary := emp\_rec.salary; -- Old salary

-- Calculate the new salary (10% increment)

v\_new\_salary := v\_old\_salary \* 1.10;

-- Update the employee's salary

UPDATE emp

SET salary = v\_new\_salary

WHERE emp\_no = emp\_rec.emp\_no;

-- Insert a record in the increment\_salary table

INSERT INTO increment\_salary (emp\_no, old\_salary, new\_salary)

VALUES (emp\_rec.emp\_no, v\_old\_salary, v\_new\_salary);

END LOOP;

-- Commit the transaction

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updates and logging completed.');

EXCEPTION

WHEN OTHERS THEN

-- Handle any errors

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

P6)

Write a **Stored Procedure** namely **proc\_Grade** for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and 900 categories is first class, if marks 899 and 825 category is Higher Second Class.

Write a PL/SQL block for using procedure created with above requirement.

Stud\_Marks(name, total\_marks),

Result (Roll,Name, Class)

-- Create table to store student marks

CREATE TABLE Stud\_Marks (

name VARCHAR2(100),

total\_marks NUMBER

);

-- Create table to store results after categorization

CREATE TABLE Result (

Roll NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Class VARCHAR2(50)

);

-- Insert sample data into Stud\_Marks

INSERT INTO Stud\_Marks (name, total\_marks) VALUES ('John Doe', 1200);

INSERT INTO Stud\_Marks (name, total\_marks) VALUES ('Jane Smith', 950);

INSERT INTO Stud\_Marks (name, total\_marks) VALUES ('Alice Johnson', 880);

INSERT INTO Stud\_Marks (name, total\_marks) VALUES ('Bob Brown', 720);

COMMIT;

CREATE OR REPLACE PROCEDURE proc\_Grade (p\_name IN VARCHAR2, p\_marks IN NUMBER) IS

v\_class VARCHAR2(50);

BEGIN

-- Determine the class based on marks

IF p\_marks <= 1500 AND p\_marks >= 990 THEN

v\_class := 'Distinction';

ELSIF p\_marks BETWEEN 900 AND 989 THEN

v\_class := 'First Class';

ELSIF p\_marks BETWEEN 825 AND 899 THEN

v\_class := 'Higher Second Class';

ELSE

v\_class := 'Fail'; -- For marks below 825

END IF;

-- Insert the result into the Result table

INSERT INTO Result (Roll, Name, Class)

VALUES (Result\_seq.NEXTVAL, p\_name, v\_class); -- Assuming a sequence is used for Roll numbers

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

-- Create a sequence for generating Roll numbers

CREATE SEQUENCE Result\_seq START WITH 1 INCREMENT BY 1;

DECLARE

v\_name VARCHAR2(100);

v\_marks NUMBER;

BEGIN

-- Loop through all students in the Stud\_Marks table

FOR stud\_rec IN (SELECT name, total\_marks FROM Stud\_Marks) LOOP

v\_name := stud\_rec.name;

v\_marks := stud\_rec.total\_marks;

-- Call the procedure to categorize the student

proc\_Grade(v\_name, v\_marks);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('Student classification completed.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

ROLLBACK;

END;

/

P7)

Create a **stored function** titled **'Age\_calc'**.

Accept the date of birth of a person as a parameter.

Calculate the age of the person in years, months and days e.g. 3 years, 2months, 10 days.

Return the age in years directly (with the help of Return statement). The months and days are to be returned indirectly in the form of OUT parameters.

CREATE OR REPLACE FUNCTION Age\_calc (

p\_dob IN DATE, -- Date of birth

p\_months OUT NUMBER, -- Out parameter for months

p\_days OUT NUMBER -- Out parameter for days

) RETURN NUMBER IS

v\_current\_date DATE := SYSDATE; -- Current date to calculate age

v\_years NUMBER; -- Variable to store years

v\_month\_diff NUMBER; -- Variable to store the month difference

v\_day\_diff NUMBER; -- Variable to store the day difference

BEGIN

-- Calculate the full years first

v\_years := TRUNC(MONTHS\_BETWEEN(v\_current\_date, p\_dob) / 12);

-- Calculate the difference in months (remaining months after full years)

v\_month\_diff := MOD(TRUNC(MONTHS\_BETWEEN(v\_current\_date, p\_dob)), 12);

-- Calculate the difference in days (remaining days after full months)

-- To calculate days, we use a "rounding" logic to ensure days are calculated correctly.

v\_day\_diff := v\_current\_date - ADD\_MONTHS(p\_dob, v\_years \* 12 + v\_month\_diff);

-- Assign the months and days values to the OUT parameters

p\_months := v\_month\_diff;

p\_days := v\_day\_diff;

-- Return the years

RETURN v\_years;

END Age\_calc;

DECLARE

v\_years NUMBER;

v\_months NUMBER;

v\_days NUMBER;

v\_dob DATE := TO\_DATE('1990-05-15', 'YYYY-MM-DD'); -- Example Date of Birth

BEGIN

-- Call the Age\_calc function

v\_years := Age\_calc(v\_dob, v\_months, v\_days);

-- Output the results

DBMS\_OUTPUT.PUT\_LINE('Age: ' || v\_years || ' years, ' || v\_months || ' months, ' || v\_days || ' days');

END;

/

P8)

Write a **Row Level Before and After Trigger** on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library\_Audit table.

CREATE TABLE Library (

book\_id NUMBER PRIMARY KEY,

title VARCHAR2(100),

author VARCHAR2(100),

publish\_year NUMBER,

genre VARCHAR2(50),

price NUMBER

);

CREATE TABLE Library\_Audit (

audit\_id NUMBER PRIMARY KEY,

book\_id NUMBER,

title VARCHAR2(100),

author VARCHAR2(100),

publish\_year NUMBER,

genre VARCHAR2(50),

price NUMBER,

action\_type VARCHAR2(10), -- 'UPDATE' or 'DELETE'

action\_timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE OR REPLACE TRIGGER Library\_Before\_Trigger

BEFORE DELETE OR UPDATE ON Library

FOR EACH ROW

BEGIN

-- If the operation is an UPDATE, store the old values before updating

IF UPDATING THEN

INSERT INTO Library\_Audit (audit\_id, book\_id, title, author, publish\_year, genre, price, action\_type)

VALUES (Library\_Audit\_SEQ.NEXTVAL, :OLD.book\_id, :OLD.title, :OLD.author, :OLD.publish\_year, :OLD.genre, :OLD.price, 'UPDATE');

END IF;

-- If the operation is a DELETE, store the old values before deleting

IF DELETING THEN

INSERT INTO Library\_Audit (audit\_id, book\_id, title, author, publish\_year, genre, price, action\_type)

VALUES (Library\_Audit\_SEQ.NEXTVAL, :OLD.book\_id, :OLD.title, :OLD.author, :OLD.publish\_year, :OLD.genre, :OLD.price, 'DELETE');

END IF;

END;

/

CREATE OR REPLACE TRIGGER Library\_After\_Trigger

AFTER DELETE OR UPDATE ON Library

FOR EACH ROW

BEGIN

-- You can log additional information here if needed after the operation

IF UPDATING THEN

DBMS\_OUTPUT.PUT\_LINE('Book updated: ' || :OLD.book\_id || ' - ' || :OLD.title);

END IF;

IF DELETING THEN

DBMS\_OUTPUT.PUT\_LINE('Book deleted: ' || :OLD.book\_id || ' - ' || :OLD.title);

END IF;

END;

/

CREATE SEQUENCE Library\_Audit\_SEQ

START WITH 1

INCREMENT BY 1;

INSERT INTO Library (book\_id, title, author, publish\_year, genre, price)

VALUES (1, 'The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Fiction', 20);

INSERT INTO Library (book\_id, title, author, publish\_year, genre, price)

VALUES (2, '1984', 'George Orwell', 1949, 'Dystopian', 15);

COMMIT;

UPDATE Library

SET price = 25

WHERE book\_id = 1;

COMMIT;

DELETE FROM Library

WHERE book\_id = 2;

COMMIT;

SELECT \* FROM Library\_Audit;

P9)

**Trigger**: Create a row level trigger for the CUSTOMERS table that would fire INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values.

CREATE TABLE CUSTOMERS (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

salary NUMBER

);

CREATE OR REPLACE TRIGGER salary\_diff\_trigger

AFTER INSERT OR UPDATE OR DELETE ON CUSTOMERS

FOR EACH ROW

DECLARE

v\_salary\_diff NUMBER;

BEGIN

-- For UPDATE operation, calculate salary difference between old and new values

IF UPDATING THEN

v\_salary\_diff := :NEW.salary - :OLD.salary;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for customer ' || :OLD.customer\_id ||

': Old Salary = ' || :OLD.salary || ', New Salary = ' || :NEW.salary ||

'. Salary Difference = ' || v\_salary\_diff);

-- For DELETE operation, display salary before deletion

ELSIF DELETING THEN

DBMS\_OUTPUT.PUT\_LINE('Salary of deleted customer ' || :OLD.customer\_id ||

': Salary = ' || :OLD.salary);

-- For INSERT operation, display the new salary

ELSIF INSERTING THEN

DBMS\_OUTPUT.PUT\_LINE('New customer added: ' || :NEW.customer\_id ||

' with Salary = ' || :NEW.salary);

END IF;

END salary\_diff\_trigger;

/

INSERT INTO CUSTOMERS (customer\_id, name, salary)

VALUES (1, 'John Doe', 5000);

COMMIT;

New customer added: 1 with Salary = 5000

UPDATE CUSTOMERS

SET salary = 5500

WHERE customer\_id = 1;

COMMIT;

Salary updated for customer 1: Old Salary = 5000, New Salary = 5500. Salary Difference = 500

DELETE FROM CUSTOMERS

WHERE customer\_id = 1;

COMMIT;

Salary of deleted customer 1: Salary = 5500

SET SERVEROUTPUT ON;

P10)

**Trigger:** Write a after trigger for Insert, update and delete event considering following requirement:

Emp(Emp\_no, Emp\_name, Emp\_salary)

1. Trigger should be initiated when salary tried to be inserted is less than Rs.50,000/-
2. Trigger should be initiated when salary tried to be updated for value less than Rs. 50,000/-

Also the new values expected to be inserted will be stored in new table Tracking(Emp\_no,Emp\_salary).

CREATE TABLE Emp (

Emp\_no NUMBER PRIMARY KEY,

Emp\_name VARCHAR2(100),

Emp\_salary NUMBER

);

CREATE TABLE Tracking (

Emp\_no NUMBER,

Emp\_salary NUMBER

);

CREATE OR REPLACE TRIGGER Emp\_salary\_tracking\_trigger

AFTER INSERT OR UPDATE OR DELETE ON Emp

FOR EACH ROW

BEGIN

-- Trigger logic for INSERT operation

IF INSERTING THEN

IF :NEW.Emp\_salary < 50000 THEN

-- Insert the new salary into Tracking table

INSERT INTO Tracking (Emp\_no, Emp\_salary)

VALUES (:NEW.Emp\_no, :NEW.Emp\_salary);

END IF;

END IF;

-- Trigger logic for UPDATE operation

IF UPDATING THEN

IF :NEW.Emp\_salary < 50000 THEN

-- Insert the updated salary into Tracking table

INSERT INTO Tracking (Emp\_no, Emp\_salary)

VALUES (:NEW.Emp\_no, :NEW.Emp\_salary);

END IF;

END IF;

-- Trigger logic for DELETE operation

IF DELETING THEN

-- Log the deleted Emp\_no in Tracking table (if needed for auditing)

INSERT INTO Tracking (Emp\_no, Emp\_salary)

VALUES (:OLD.Emp\_no, NULL); -- Store NULL for salary on delete

END IF;

END Emp\_salary\_tracking\_trigger;

/

INSERT INTO Emp (Emp\_no, Emp\_name, Emp\_salary)

VALUES (101, 'John Doe', 40000);

COMMIT;

SELECT \* FROM Tracking;

UPDATE Emp

SET Emp\_salary = 45000

WHERE Emp\_no = 101;

COMMIT;

SELECT \* FROM Tracking;

DELETE FROM Emp

WHERE Emp\_no = 101;

COMMIT;

SELECT \* FROM Tracking;

**Mongodb:-**

M1)

**Design and Develop MongoDB Queries using CRUD operations:**

Create Employee collection by considering following Fields:

1. Name: Embedded Doc (FName, LName)
2. Company Name: String
3. Salary: Number
4. Designation: String
5. Age: Number
6. Expertise: Array
7. DOB: String or Date
8. Email id: String ix. Contact: String

x. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above attribute and execute following queries:

1. Select all documents where the Designation field has the value "Programmer" and the value of the salary field is greater than 30000.
2. Creates a new document if no document in the employee collection contains

{Designation: "Tester", Company\_name: "TCS", Age: 25}

1. Increase salary of each Employee working with “Infosys" 10000.
2. Finds all employees working with "TCS" and reduce their salary by 5000.
3. Return documents where Designation is not equal to "Tester".
4. Find all employee with Exact Match on an Array having Expertise:

['Mongodb','Mysql','Cassandra']

db.createCollection("Employee");

db.Employee.insertMany([

{

Name: { FName: "John", LName: "Doe" },

Company\_Name: "TCS",

Salary: 50000,

Designation: "Programmer",

Age: 30,

Expertise: ["Java", "MongoDB", "Spring"],

DOB: new Date("1994-05-15"),

Email: "john.doe@example.com",

Contact: "9876543210",

Address: [{ PAddr: "123 Main St, CityA", LAddr: "456 Elm St, CityA" }]

},

{

Name: { FName: "Jane", LName: "Smith" },

Company\_Name: "Infosys",

Salary: 70000,

Designation: "Senior Programmer",

Age: 28,

Expertise: ["JavaScript", "Node.js", "AWS"],

DOB: new Date("1996-02-22"),

Email: "jane.smith@example.com",

Contact: "9876543211",

Address: [{ PAddr: "789 Pine St, CityB", LAddr: "101 Maple St, CityB" }]

},

{

Name: { FName: "Sam", LName: "Wilson" },

Company\_Name: "TCS",

Salary: 35000,

Designation: "Programmer",

Age: 27,

Expertise: ["Python", "Django", "MySQL"],

DOB: new Date("1997-11-30"),

Email: "sam.wilson@example.com",

Contact: "9876543212",

Address: [{ PAddr: "101 Oak St, CityC", LAddr: "202 Birch St, CityC" }]

},

{

Name: { FName: "Alice", LName: "Johnson" },

Company\_Name: "Infosys",

Salary: 80000,

Designation: "Tester",

Age: 35,

Expertise: ["Manual Testing", "Automation Testing", "Selenium"],

DOB: new Date("1989-07-10"),

Email: "alice.johnson@example.com",

Contact: "9876543213",

Address: [{ PAddr: "123 River St, CityD", LAddr: "456 Park St, CityD" }]

},

{

Name: { FName: "David", LName: "Brown" },

Company\_Name: "Wipro",

Salary: 60000,

Designation: "Programmer",

Age: 32,

Expertise: ["Java", "Hibernate", "PostgreSQL"],

DOB: new Date("1992-12-20"),

Email: "david.brown@example.com",

Contact: "9876543214",

Address: [{ PAddr: "567 Lake St, CityE", LAddr: "789 Cedar St, CityE" }]

}

]);

db.Employee.find({

Designation: "Programmer",

Salary: { $gt: 30000 }

});

db.Employee.updateOne(

{ Designation: "Tester", Company\_Name: "TCS", Age: 25 },

{ $setOnInsert: {

Name: { FName: "New", LName: "Tester" },

Company\_Name: "TCS",

Salary: 35000,

Designation: "Tester",

Age: 25,

Expertise: ["Manual Testing", "Selenium"],

DOB: new Date("1999-01-01"),

Email: "new.tester@example.com",

Contact: "9876543215",

Address: [{ PAddr: "Unknown St, CityF", LAddr: "Unknown St, CityF" }]

}},

{ upsert: true }

);

db.Employee.updateMany(

{ Company\_Name: "Infosys" },

{ $inc: { Salary: 10000 } }

);

db.Employee.updateMany(

{ Company\_Name: "TCS" },

{ $inc: { Salary: -5000 } }

);

db.Employee.find({

Designation: { $ne: "Tester" }

});

db.Employee.find({

Expertise: { $eq: ['Mongodb', 'Mysql', 'Cassandra'] }

});

M2)

**Design and Develop MongoDB Queries using CRUD operations:**

Create Employee collection by considering following Fields:

1. Name: Embedded Doc (FName, LName)
2. Company Name: String
3. Salary: Number
4. Designation: String
5. Age: Number
6. Expertise: Array
7. DOB: String or Date
8. Email id: String ix. Contact: String

x. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above attribute and execute following queries:

1. Final name of Employee where age is less than 30 and salary more than 50000.
2. Creates a new document if no document in the employee collection contains

{Designation: "Tester", Company\_name: "TCS", Age: 25}

1. Selects all documents in the collection where the field age has a value less than 30 or the value of the salary field is greater than 40000.
2. Find documents where Designation is not equal to "Developer".
3. Find \_id, Designation, Address and Name from all documents where Company\_name is "Infosys".
4. Display only FName and LName of all Employees

db.Employee.insertMany([

{

Name: { FName: "John", LName: "Doe" },

Company\_Name: "TCS",

Salary: 60000,

Designation: "Programmer",

Age: 28,

Expertise: ["Java", "MongoDB", "Spring"],

DOB: new Date("1996-05-15"),

Email: "john.doe@example.com",

Contact: "9876543210",

Address: [{ PAddr: "123 Main St, CityA", LAddr: "456 Elm St, CityA" }]

},

{

Name: { FName: "Jane", LName: "Smith" },

Company\_Name: "Infosys",

Salary: 75000,

Designation: "Senior Developer",

Age: 35,

Expertise: ["JavaScript", "React", "Node.js"],

DOB: new Date("1989-04-22"),

Email: "jane.smith@example.com",

Contact: "9876543211",

Address: [{ PAddr: "789 Pine St, CityB", LAddr: "101 Maple St, CityB" }]

},

{

Name: { FName: "Sam", LName: "Wilson" },

Company\_Name: "Infosys",

Salary: 50000,

Designation: "Developer",

Age: 27,

Expertise: ["Python", "Django", "MySQL"],

DOB: new Date("1997-11-30"),

Email: "sam.wilson@example.com",

Contact: "9876543212",

Address: [{ PAddr: "101 Oak St, CityC", LAddr: "202 Birch St, CityC" }]

},

{

Name: { FName: "Alice", LName: "Johnson" },

Company\_Name: "Wipro",

Salary: 45000,

Designation: "Tester",

Age: 25,

Expertise: ["Manual Testing", "Selenium", "Automation"],

DOB: new Date("1999-07-10"),

Email: "alice.johnson@example.com",

Contact: "9876543213",

Address: [{ PAddr: "123 River St, CityD", LAddr: "456 Park St, CityD" }]

},

{

Name: { FName: "David", LName: "Brown" },

Company\_Name: "TCS",

Salary: 70000,

Designation: "Programmer",

Age: 32,

Expertise: ["Java", "Hibernate", "PostgreSQL"],

DOB: new Date("1992-12-20"),

Email: "david.brown@example.com",

Contact: "9876543214",

Address: [{ PAddr: "567 Lake St, CityE", LAddr: "789 Cedar St, CityE" }]

}

]);

db.Employee.find(

{

Age: { $lt: 30 },

Salary: { $gt: 50000 }

},

{ "Name": 1, "\_id": 0 } // Only return Name and exclude the \_id

);

db.Employee.updateOne(

{ Designation: "Tester", Company\_Name: "TCS", Age: 25 },

{

$setOnInsert: {

Name: { FName: "New", LName: "Tester" },

Company\_Name: "TCS",

Salary: 35000,

Designation: "Tester",

Age: 25,

Expertise: ["Manual Testing", "Selenium"],

DOB: new Date("1999-01-01"),

Email: "new.tester@example.com",

Contact: "9876543215",

Address: [{ PAddr: "Unknown St, CityF", LAddr: "Unknown St, CityF" }]

}

},

{ upsert: true } // Only insert if no document matches

);

db.Employee.find({

$or: [

{ Age: { $lt: 30 } },

{ Salary: { $gt: 40000 } }

]

});

db.Employee.find({

Designation: { $ne: "Developer" }

});

db.Employee.find(

{ Company\_Name: "Infosys" },

{ \_id: 1, Designation: 1, Address: 1, Name: 1 }

);

db.Employee.find({}, { "Name.FName": 1, "Name.LName": 1, "\_id": 0 });

M3)

**Design and Develop MongoDB Queries using CRUD operations:**

Create Employee collection by considering following Fields: i. Emp\_id : Number

1. Name: Embedded Doc (FName, LName)
2. Company Name: String
3. Salary: Number
4. Designation: String
5. Age: Number
6. Expertise: Array
7. DOB: String or Date
8. Email id: String x. Contact: String

xi. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above attribute and execute following queries:

1. Creates a new document if no document in the employee collection contains

{Designation: "Tester", Company\_name: "TCS", Age: 25}

1. Finds all employees working with Company\_name: "TCS" and increase their salary by 2000.
2. Matches all documents where the value of the field Address is an embedded document that contains only the field city with the value "Pune" and the field Pin\_code with the value "411001".
3. Find employee details who are working as "Developer" or "Tester".
4. Drop Single documents where designation="Developer".
5. Count number of documents in employee collection.

db.Employee.updateOne(

{ Designation: "Tester", Company\_Name: "TCS", Age: 25 },

{

$setOnInsert: {

Name: { FName: "New", LName: "Tester" },

Company\_Name: "TCS",

Salary: 35000,

Designation: "Tester",

Age: 25,

Expertise: ["Manual Testing", "Selenium"],

DOB: new Date("1999-01-01"),

Email: "new.tester@example.com",

Contact: "9876543215",

Address: [{ PAddr: "Unknown St, CityF", LAddr: "Unknown St, CityF", City: "Pune", Pin\_code: "411001" }]

}

},

{ upsert: true } // Insert if no document matches the query

);

db.Employee.updateMany(

{ Company\_Name: "TCS" },

{ $inc: { Salary: 2000 } }

);

db.Employee.find({

"Address.City": "Pune",

"Address.Pin\_code": "411001"

});

db.Employee.find({

Designation: { $in: ["Developer", "Tester"] }

});

db.Employee.deleteOne({

Designation: "Developer"

});

db.Employee.countDocuments();

M4)

**Design and Develop MongoDB Queries using Aggregation operations:**

Create Employee collection by considering following Fields: i. Emp\_id : Number

1. Name: Embedded Doc (FName, LName)
2. Company Name: String
3. Salary: Number
4. Designation: String
5. Age: Number
6. Expertise: Array
7. DOB: String or Date
8. Email id: String x. Contact: String

xi. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above attribute and execute following:

1. Using aggregation Return Designation with Total Salary is Above 200000.
2. Using Aggregate method returns names and \_id in upper case and in alphabetical order.
3. Using aggregation method find Employee with Total Salary for Each City with Designation="DBA".
4. Create Single Field Indexes on Designation field of employee collection
5. To Create Multikey Indexes on Expertise field of employee collection.
6. Create an Index on Emp\_id field, compare the time require to search Emp\_id before and after creating an index. (Hint Add at least 10000 Documents)
7. Return a List of Indexes on created on employee Collection.

db.Employee.aggregate([

{

$group: {

\_id: "$Designation",

totalSalary: { $sum: "$Salary" }

}

},

{

$match: {

totalSalary: { $gt: 200000 }

}

}

]);

db.Employee.aggregate([

{

$project: {

\_id: 1,

Name: {

FName: { $toUpper: "$Name.FName" },

LName: { $toUpper: "$Name.LName" }

}

}

},

{

$sort: { "Name.FName": 1, "Name.LName": 1 }

}

]);

db.Employee.aggregate([

{

$match: { Designation: "DBA" }

},

{

$group: {

\_id: "$Address.City",

totalSalary: { $sum: "$Salary" }

}

}

]);

db.Employee.createIndex({ Designation: 1 });

db.Employee.createIndex({ Expertise: 1 });

// Insert 10,000 sample documents (for the sake of example)

for (let i = 0; i < 10000; i++) {

db.Employee.insertOne({

Emp\_id: i,

Name: { FName: `Employee${i}`, LName: `Lastname${i}` },

Company\_Name: "Company" + i % 3,

Salary: 50000 + (i % 5000),

Designation: i % 2 === 0 ? "Programmer" : "Tester",

Age: 30 + (i % 10),

Expertise: ["Java", "MongoDB"],

DOB: new Date("1990-01-01"),

Email: `employee${i}@example.com`,

Contact: `98765432${i}`,

Address: [{ PAddr: "Address", LAddr: "Local Address", City: "City", Pin\_code: "411001" }]

});

}

// Now, create an index on the `Emp\_id` field

db.Employee.createIndex({ Emp\_id: 1 });

// Query before creating the index (without index)

let start = new Date();

db.Employee.find({ Emp\_id: 5000 }).toArray(); // You can choose any Emp\_id

let end = new Date();

print(`Time taken without index: ${end - start} ms`);

// Query after creating the index (with index)

start = new Date();

db.Employee.find({ Emp\_id: 5000 }).toArray(); // You can choose any Emp\_id

end = new Date();

print(`Time taken with index: ${end - start} ms`);

db.Employee.getIndexes();

M5)

**Design and Develop MongoDB Queries using Aggregation operations:**

Create Employee collection by considering following Fields: i. Emp\_id : Number

1. Name: Embedded Doc (FName, LName)
2. Company Name: String
3. Salary: Number
4. Designation: String
5. Age: Number
6. Expertise: Array
7. DOB: String or Date
8. Email id: String x. Contact: String

xi. Address: Array of Embedded Doc (PAddr, LAddr)

Insert at least 5 documents in collection by considering above attribute and execute following:

1. Using aggregation Return separates value in the Expertise array and return sum of each element of array.
2. Using Aggregate method return Max and Min Salary for each company.
3. Using Aggregate method find Employee with Total Salary for Each City with Designation="DBA".
4. Using aggregation method Return separates value in the Expertise array for employee name where Swapnil Jadhav
5. To Create Compound Indexes on Name: 1, Age: -1
6. Create an Index on Emp\_id field, compare the time require to search Emp\_id before and after creating an index. (Hint Add at least 10000 Documents)
7. Return a List of Indexes on created on employee Collection.

db.Employee.insertMany([

{

Name: { FName: "John", LName: "Doe" },

Company\_Name: "TCS",

Salary: 60000,

Designation: "Programmer",

Age: 28,

Expertise: ["Java", "MongoDB", "Spring"],

DOB: new Date("1996-05-15"),

Email: "john.doe@example.com",

Contact: "9876543210",

Address: [{ PAddr: "123 Main St", LAddr: "456 Elm St", City: "Pune", Pin\_code: "411001" }]

},

{

Name: { FName: "Jane", LName: "Smith" },

Company\_Name: "Infosys",

Salary: 75000,

Designation: "Senior Developer",

Age: 35,

Expertise: ["JavaScript", "React", "Node.js"],

DOB: new Date("1989-04-22"),

Email: "jane.smith@example.com",

Contact: "9876543211",

Address: [{ PAddr: "789 Pine St", LAddr: "101 Maple St", City: "Bangalore", Pin\_code: "560001" }]

},

{

Name: { FName: "Swapnil", LName: "Jadhav" },

Company\_Name: "TCS",

Salary: 50000,

Designation: "DBA",

Age: 30,

Expertise: ["MongoDB", "SQL", "Database Administration"],

DOB: new Date("1994-11-30"),

Email: "swapnil.jadhav@example.com",

Contact: "9876543212",

Address: [{ PAddr: "101 Oak St", LAddr: "202 Birch St", City: "Pune", Pin\_code: "411001" }]

},

{

Name: { FName: "Alice", LName: "Johnson" },

Company\_Name: "Wipro",

Salary: 45000,

Designation: "Tester",

Age: 25,

Expertise: ["Manual Testing", "Selenium", "Automation"],

DOB: new Date("1999-07-10"),

Email: "alice.johnson@example.com",

Contact: "9876543213",

Address: [{ PAddr: "123 River St", LAddr: "456 Park St", City: "Pune", Pin\_code: "411001" }]

},

{

Name: { FName: "David", LName: "Brown" },

Company\_Name: "TCS",

Salary: 70000,

Designation: "DBA",

Age: 32,

Expertise: ["SQL", "Database Management", "PostgreSQL"],

DOB: new Date("1992-12-20"),

Email: "david.brown@example.com",

Contact: "9876543214",

Address: [{ PAddr: "567 Lake St", LAddr: "789 Cedar St", City: "Pune", Pin\_code: "411001" }]

}

]);

db.Employee.aggregate([

{ $unwind: "$Expertise" },

{

$group: {

\_id: "$Expertise",

count: { $sum: 1 }

}

}

]);

db.Employee.aggregate([

{

$group: {

\_id: "$Company\_Name",

maxSalary: { $max: "$Salary" },

minSalary: { $min: "$Salary" }

}

}

]);

db.Employee.aggregate([

{ $match: { Designation: "DBA" } },

{

$group: {

\_id: "$Address.City",

totalSalary: { $sum: "$Salary" }

}

}

]);

db.Employee.aggregate([

{ $match: { "Name.FName": "Swapnil", "Name.LName": "Jadhav" } },

{ $unwind: "$Expertise" },

{ $project: { Expertise: 1 } }

]);

db.Employee.createIndex({ "Name.FName": 1, "Age": -1 });

// Insert 10,000 sample documents (for testing)

for (let i = 0; i < 10000; i++) {

db.Employee.insertOne({

Emp\_id: i,

Name: { FName: `Employee${i}`, LName: `Lastname${i}` },

Company\_Name: "Company" + (i % 3),

Salary: 50000 + (i % 5000),

Designation: i % 2 === 0 ? "Programmer" : "Tester",

Age: 30 + (i % 10),

Expertise: ["Java", "MongoDB"],

DOB: new Date("1990-01-01"),

Email: `employee${i}@example.com`,

Contact: `98765432${i}`,

Address: [{ PAddr: "Address", LAddr: "Local Address", City: "City", Pin\_code: "411001" }]

});

}

// Create an index on Emp\_id

db.Employee.createIndex({ Emp\_id: 1 });

// Measure time taken to find Emp\_id without index

let start = new Date();

db.Employee.find({ Emp\_id: 5000 }).toArray(); // Search for Emp\_id = 5000

let end = new Date();

print(`Time taken without index: ${end - start} ms`);

// Measure time taken to find Emp\_id with index

start = new Date();

db.Employee.find({ Emp\_id: 5000 }).toArray(); // Search for Emp\_id = 5000

end = new Date();

print(`Time taken with index: ${end - start} ms`);

db.Employee.getIndexes();

M6)

**Design MongoDB database and perform following Map reduce operation:**

Create Employee collection by considering following Fields:

1. Name: Embedded Doc (FName, LName)
2. Company Name: String
3. Salary: Number
4. Designation: String
5. Age: Number
6. Expertise: Array
7. DOB: String or Date
8. Email id: String ix. Contact: String

x. Address: Array of Embedded Doc (PAddr, LAddr) Execute the following query:

1. Display the total salary of per company
2. Display the total salary of company Name:"TCS"
3. Return the average salary of company whose address is “Pune".
4. Display total count for “City=Pune”

db.Employee.aggregate([

{

$group: {

\_id: "$Company\_Name",

totalSalary: { $sum: "$Salary" }

}

}

]);

db.Employee.aggregate([

{ $match: { Company\_Name: "TCS" } },

{

$group: {

\_id: "$Company\_Name",

totalSalary: { $sum: "$Salary" }

}

}

]);

db.Employee.aggregate([

{ $match: { "Address.City": "Pune" } },

{

$group: {

\_id: null,

averageSalary: { $avg: "$Salary" }

}

}

]);

db.Employee.aggregate([

{ $match: { "Address.City": "Pune" } },

{ $count: "totalCount" }

]);

db.Employee.aggregate([

{ $match: { "Address.City": "Pune", Age: { $gt: 40 } } },

{ $count: "totalCount" }

]);