**1) Consider the relation employee (emp\_id,e\_name,salary ,Date of Joining,Dapt\_no,Designation) perform basic SQL operations.**

1. **Create table employee.**
2. **Insert 10 records in table.**
3. **Create a view emp\_vl of table employee which has emp\_id , name and dept-attributes.**
4. **Create view of table.**
5. **Update dept of any employee in view. Check whether it gets updated or not.**
6. **Create emp\_id as primary key and show indices on table employee.**
7. **Show indices on table.**
8. **Create user defined index on any column.**

Ans :

CREATE TABLE employee (

emp\_id INT,

e\_name VARCHAR(50),

salary DECIMAL(10, 2),

date\_of\_joining DATE,

dapt\_no INT,

designation VARCHAR(50)

);

INSERT INTO employee (emp\_id, e\_name, salary, date\_of\_joining, dapt\_no, designation) VALUES

(1, 'Alice', 50000, '2020-01-15', 101, 'Manager'),

(2, 'Bob', 40000, '2021-02-20', 102, 'Analyst'),

(3, 'Charlie', 45000, '2019-03-10', 103, 'Developer'),

(4, 'David', 55000, '2018-04-18', 104, 'Designer'),

(5, 'Eve', 60000, '2020-05-22', 101, 'Manager'),

(6, 'Frank', 37000, '2021-06-25', 102, 'Analyst'),

(7, 'Grace', 43000, '2019-07-30', 103, 'Developer'),

(8, 'Heidi', 48000, '2018-08-15', 104, 'Designer'),

(9, 'Ivy', 52000, '2020-09-10', 101, 'Manager'),

(10, 'Jack', 41000, '2021-10-22', 102, 'Analyst');

CREATE VIEW emp\_v1 AS

SELECT emp\_id, e\_name, dapt\_no

FROM employee;

SELECT \* FROM emp\_v1;

UPDATE emp\_v1

SET dapt\_no = 105

WHERE emp\_id = 2;

ALTER TABLE employee

ADD PRIMARY KEY (emp\_id);

CREATE INDEX idx\_salary ON employee(salary);

**2) Consider the relation employee (emp\_id,e\_name,salary ,Date of Joining,Dapt\_no,Designation) perform basic SQL operations.**

1. **Display employees whose name contains letter ‘e’.**
2. **Display different types of designation**
3. **Display name and salary of employee whose location is Mumbai**
4. **Display name and department of employee working in Manager or Marketing department**
5. **Display the department name whose employees are more than one**
6. **Rename employee table as emp1**
7. **Add a new column city in the employee table.**

Ans :

SELECT \* FROM employee

WHERE e\_name LIKE '%e%';

SELECT DISTINCT designation

FROM employee;

SELECT e\_name, salary

FROM employee

WHERE city = 'Mumbai';

SELECT e\_name, dapt\_no

FROM employee

WHERE designation IN ('Manager', 'Marketing');

SELECT dapt\_no, COUNT(emp\_id) AS employee\_count

FROM employee

GROUP BY dapt\_no

HAVING COUNT(emp\_id) > 1;

RENAME TABLE employee TO emp1;

ALTER TABLE employee

ADD COLUMN city VARCHAR(50);

**3)Consider the relation employee(emp\_id,e\_name,salary ,Date of Joining,Dapt\_no,Designation) perform basic SQL operations.**

1. **Find department in which maximum employees work.**
2. **Display name, designation and department no of employees whose name starts with either ‘A’ or ‘P’.**
3. **Display max. salary from department 2 and min. salary from department 4.**
4. **Display employee data where salary is less than average salary from department 3.**
5. **Display employees who were hired earliest or latest.**
6. **Display name and department no of employees who are manager, market analysts. Use prediactes**
7. **List employees hired in August.**
8. **List employees who are hired after 31/12/2006.**
9. **Find average annual salary per department.**

Ans :

SELECT dapt\_no, COUNT(emp\_id) AS employee\_count

FROM employee

GROUP BY dapt\_no

ORDER BY employee\_count DESC

LIMIT 1;

SELECT e\_name, designation, dapt\_no

FROM employee

WHERE e\_name LIKE 'A%' OR e\_name LIKE 'P%';

SELECT MAX(salary) AS max\_salary\_dept2

FROM employee

WHERE dapt\_no = 2;

SELECT MIN(salary) AS min\_salary\_dept4

FROM employee

WHERE dapt\_no = 4;

SELECT \*

FROM employee

WHERE salary < (SELECT AVG(salary) FROM employee WHERE dapt\_no = 3);

SELECT \*

FROM employee

WHERE Date\_of\_Joining = (SELECT MIN(Date\_of\_Joining) FROM employee)

OR Date\_of\_Joining = (SELECT MAX(Date\_of\_Joining) FROM employee);

SELECT e\_name, dapt\_no

FROM employee

WHERE designation IN ('Manager', 'Market Analyst');

SELECT \*

FROM employee

WHERE MONTH(Date\_of\_Joining) = 8;

SELECT \*

FROM employee

WHERE Date\_of\_Joining > '2006-12-31';

SELECT dapt\_no, AVG(salary) AS avg\_annual\_salary

FROM employee

GROUP BY dapt\_no;

**4)Consider two tables Customer(c\_id, c\_name , email , city , pincode)Order(order\_id , date , amount , cust\_id.**

1. **Create both the tables with primary key and foreign key constraints.**
2. **insert 10 records each.**
3. **Find all orders placed by customers with cust\_id 2**
4. **Find list of customers who placed their order and details of order**
5. **List of customers who haven’t placed order**
6. **List all orders and append to customer table**
7. **Display all records**
8. **Display customer that are from same city8**

Ans :

CREATE TABLE Customer (

c\_id INT PRIMARY KEY,

c\_name VARCHAR(50),

email VARCHAR(50),

city VARCHAR(50),

pincode VARCHAR(6)

);

CREATE TABLE Order (

order\_id INT PRIMARY KEY,

date DATE,

amount DECIMAL(10, 2),

cust\_id INT,

FOREIGN KEY (cust\_id) REFERENCES Customer(c\_id)

);

INSERT INTO Customer (c\_id, c\_name, email, city, pincode) VALUES

(1, 'Alice', 'alice@example.com', 'Mumbai', '400001'),

(2, 'Bob', 'bob@example.com', 'Delhi', '110001'),

(3, 'Charlie', 'charlie@example.com', 'Mumbai', '400002'),

(4, 'David', 'david@example.com', 'Chennai', '600001'),

(5, 'Eve', 'eve@example.com', 'Bangalore', '560001'),

(6, 'Frank', 'frank@example.com', 'Delhi', '110002'),

(7, 'Grace', 'grace@example.com', 'Mumbai', '400003'),

(8, 'Heidi', 'heidi@example.com', 'Kolkata', '700001'),

(9, 'Ivy', 'ivy@example.com', 'Bangalore', '560002'),

(10, 'Jack', 'jack@example.com', 'Chennai', '600002');

INSERT INTO Order (order\_id, date, amount, cust\_id) VALUES

(101, '2023-01-10', 500.00, 2),

(102, '2023-02-15', 1000.00, 3),

(103, '2023-03-20', 750.00, 5),

(104, '2023-04-25', 200.00, 2),

(105, '2023-05-05', 1200.00, 7),

(106, '2023-06-10', 300.00, 1),

(107, '2023-07-15', 400.00, 6),

(108, '2023-08-20', 550.00, 9),

(109, '2023-09-25', 800.00, 10),

(110, '2023-10-30', 1500.00, 3);

SELECT \* FROM Order

WHERE cust\_id = 2;

SELECT Customer.c\_id, Customer.c\_name, Order.order\_id, Order.date, Order.amount

FROM Customer

JOIN Order ON Customer.c\_id = Order.cust\_id;

SELECT Customer.c\_id, Customer.c\_name

FROM Customer

LEFT JOIN Order ON Customer.c\_id = Order.cust\_id

WHERE Order.order\_id IS NULL;

SELECT Customer.\*, Order.order\_id, Order.date, Order.amount

FROM Customer

LEFT JOIN Order ON Customer.c\_id = Order.cust\_id;

-- Display all customers

SELECT \* FROM Customer;

-- Display all orders

SELECT \* FROM Order;

SELECT city, GROUP\_CONCAT(c\_name) AS customers

FROM Customer

GROUP BY city

HAVING COUNT(c\_id) > 1;

**5) Consider tables Borrower (RollNo, Name, DateofIssue, NameofBook, Status) and**

**Fine (Roll\_no,Date,Amt). Status is either Issued or Returned.**

**1. Create both the tables with primary key.**

**2. Insert 10 records each.**

**3. Find count of books with Issued status.**

**4. Display all records.**

**5. Display RollNo whose date of issue is same.**

Ans :

CREATE TABLE Borrower (

RollNo INT PRIMARY KEY,

Name VARCHAR(50),

DateofIssue DATE,

NameofBook VARCHAR(100),

Status VARCHAR(10) CHECK (Status IN ('Issued', 'Returned'))

);

CREATE TABLE Fine (

Roll\_no INT,

Date DATE,

Amt DECIMAL(10, 2),

PRIMARY KEY (Roll\_no, Date)

);

INSERT INTO Borrower (RollNo, Name, DateofIssue, NameofBook, Status) VALUES

(1, 'Alice', '2023-01-05', 'Mathematics', 'Issued'),

(2, 'Bob', '2023-02-10', 'Physics', 'Returned'),

(3, 'Charlie', '2023-03-15', 'Chemistry', 'Issued'),

(4, 'David', '2023-04-20', 'Biology', 'Returned'),

(5, 'Eve', '2023-05-25', 'Mathematics', 'Issued'),

(6, 'Frank', '2023-06-10', 'Physics', 'Returned'),

(7, 'Grace', '2023-07-15', 'Chemistry', 'Issued'),

(8, 'Heidi', '2023-08-20', 'Biology', 'Returned'),

(9, 'Ivy', '2023-09-25', 'Mathematics', 'Issued'),

(10, 'Jack', '2023-10-30', 'Physics', 'Issued');

INSERT INTO Fine (Roll\_no, Date, Amt) VALUES

(1, '2023-01-20', 10.00),

(2, '2023-02-15', 5.00),

(3, '2023-03-25', 15.00),

(4, '2023-04-30', 8.00),

(5, '2023-05-30', 10.00),

(6, '2023-06-20', 7.00),

(7, '2023-07-25', 12.00),

(8, '2023-08-30', 6.00),

(9, '2023-09-30', 9.00),

(10, '2023-10-31', 11.00);

SELECT COUNT(\*) AS IssuedBooksCount

FROM Borrower

WHERE Status = 'Issued';

-- Display all records from Borrower table

SELECT \* FROM Borrower;

-- Display all records from Fine table

SELECT \* FROM Fine;

SELECT DateofIssue, GROUP\_CONCAT(RollNo) AS RollNos

FROM Borrower

GROUP BY DateofIssue

HAVING COUNT(RollNo) > 1;

**6) Consider student (roll\_no, name, marks, class) table. Column roll\_no is primary key. Perform any 3 DLL and any 3 DML operations on the table.**

Ans :

CREATE TABLE student (

roll\_no INT PRIMARY KEY,

name VARCHAR(50),

marks INT,

class VARCHAR(10)

);

ALTER TABLE student

ADD COLUMN age INT;

ALTER TABLE student

MODIFY marks INT CHECK (marks >= 0);

INSERT INTO student (roll\_no, name, marks, class, age) VALUES

(1, 'Alice', 85, '10A', 15),

(2, 'Bob', 78, '10B', 16),

(3, 'Charlie', 92, '10A', 15);

UPDATE student

SET marks = 88

WHERE roll\_no = 2;

DELETE FROM student

WHERE roll\_no = 3;

**7) Write a SQL statement to create a table job\_history including columns employee\_id, start\_date, end\_date, job\_id and department\_id and make sure that, the employee\_id column does not contain any duplicate value at the time of insertion and the foreign key column job\_id contain only those values which are exists in the jobs table. Consider table Job (job\_id,job\_title.min\_sal,max\_sal)**

Ans :

CREATE TABLE job\_history (

employee\_id INT UNIQUE,

start\_date DATE,

end\_date DATE,

job\_id INT,

department\_id INT,

FOREIGN KEY (job\_id) REFERENCES Job(job\_id)

);

**8) For the given relation schema: employee(employee-name, street, city)**

**works (employee-name, company-name, salary)**

**company (company-name, city)**

**manages (employee-name, manager-name)**

**Give an expression in SQL for each of the following queries:**

**a) Find the names, street address, and cities of residence for all employees who work for same company and earn more than $10,000.**

**b) Find the names of all employees in the database who live in the same cities as the companies for which they work.**

**c) Find the names of all employees who earn more than the average salary of all employees of their company. Assume that all people work for at most one company.**

Ans :

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

FROM employee

JOIN works ON employee.employee\_name = works.employee\_name

WHERE works.salary > 10000;

SELECT employee.employee\_name

FROM employee

JOIN works ON employee.employee\_name = works.employee\_name

JOIN company ON works.company\_name = company.company\_name

WHERE employee.city = company.city;

SELECT works.employee\_name

FROM works

WHERE works.salary > (

SELECT AVG(salary)

FROM works AS w

WHERE w.company\_name = works.company\_name

);

**9) For the given relation schema: employee(employee-name, street, city)**

**works (employee-name, company-name, salary)**

**company (company-name, city)**

**manages (employee-name, manager-name)**

**Give an expression in SQL for each of the following queries:**

1. **Find the name of the company that has the smallest payroll.**
2. **Find the names of all employees in the database who live in the same cities and on the same streets as do their managers.**

Ans :

SELECT company\_name

FROM works

GROUP BY company\_name

ORDER BY SUM(salary) ASC

LIMIT 1;

SELECT e.employee\_name

FROM employee e

JOIN manages m ON e.employee\_name = m.employee\_name

JOIN employee manager ON m.manager\_name = manager.employee\_name

WHERE e.street = manager.street AND e.city = manager.city;

**Mongodb**

10) Implement CRUD operations. SAVE method. Use following Collection. Perform Map Reduce to count quantity of each item.

Item: Item ID, Item quantity, price, brand.

Ans :

// Connect to MongoDB and use the appropriate database

use your\_database\_name;

// Create the items collection and insert sample documents

db.items.insertMany([

{ item\_id: 1, item\_quantity: 10, price: 5.0, brand: "BrandA" },

{ item\_id: 2, item\_quantity: 15, price: 7.5, brand: "BrandB" },

{ item\_id: 3, item\_quantity: 5, price: 10.0, brand: "BrandA" },

{ item\_id: 4, item\_quantity: 8, price: 12.0, brand: "BrandC" },

{ item\_id: 5, item\_quantity: 20, price: 6.0, brand: "BrandB" }

]);

// Create a new item

db.items.insertOne({ item\_id: 6, item\_quantity: 12, price: 8.0, brand: "BrandD" });

// Read all items

db.items.find();

// Read a specific item by item\_id

db.items.findOne({ item\_id: 2 });

// Update the quantity of an item

db.items.updateOne(

{ item\_id: 3 }, // Filter

{ $set: { item\_quantity: 7 } } // Update operation

);

// Delete an item by item\_id

db.items.deleteOne({ item\_id: 4 });

var mapFunction = function() {

emit(this.item\_id, this.item\_quantity);

};

var reduceFunction = function(key, values) {

return Array.sum(values);

};

db.items.mapReduce(

mapFunction,

reduceFunction,

{

out: "item\_quantity\_count" // Output collection

}

);

db.item\_quantity\_count.find();

**11) Implement CRUD operations. SAVE method. Use following Collection.**

**Item: Item ID, Item quantity, price, brand.**

12) **Implement CRUD operations. SAVE method. Use following Collection.**

**Item: Item ID, Item quantity, price, brand, discount**

**1. Display the count of item brand wise.**

**2. Dsiplay item with minimum price.**

**3. Display maximum discount given for item.**

Ans :

db.items.aggregate([

{

$group: {

\_id: "$brand", // Group by brand

count: { $sum: 1 } // Count the number of items per brand

}

}

]);

db.items.find().sort({ price: 1 }).limit(1);

db.items.aggregate([

{

$group: {

\_id: null, // No grouping key

maxDiscount: { $max: "$discount" } // Find maximum discount

}

}

]);

**13) Implement Map reduces operation for counting the marks of students.**

**Use: student (roll\_no, name marks, class)**

**Expected output: student name or roll no and total marks.**

Ans :

// Connect to your MongoDB instance and use your database

use your\_database\_name;

// Create the students collection and insert sample documents

db.students.insertMany([

{ roll\_no: 1, name: "Alice", marks: 85, class: "10th" },

{ roll\_no: 2, name: "Bob", marks: 90, class: "10th" },

{ roll\_no: 3, name: "Charlie", marks: 75, class: "10th" },

{ roll\_no: 4, name: "David", marks: 88, class: "10th" },

{ roll\_no: 5, name: "Eve", marks: 95, class: "10th" }

]);

var mapFunction = function() {

emit(this.roll\_no, this.marks); // Emitting roll\_no as the key and marks as the value

};

var reduceFunction = function(key, values) {

return Array.sum(values); // Summing up the marks for the same roll\_no

};

db.students.mapReduce(

mapFunction,

reduceFunction,

{

out: "student\_total\_marks" // Output collection for the results

}

);

db.student\_total\_marks.find();

**14) Implement Map reduces operation for displaying persons with same profession.**

**Use: person (person\_id, name, addr, profession)**

Ans :

// Connect to your MongoDB instance and use your database

use your\_database\_name;

// Create the person collection and insert sample documents

db.person.insertMany([

{ person\_id: 1, name: "Alice", addr: "123 Main St", profession: "Engineer" },

{ person\_id: 2, name: "Bob", addr: "456 Maple Ave", profession: "Doctor" },

{ person\_id: 3, name: "Charlie", addr: "789 Elm St", profession: "Engineer" },

{ person\_id: 4, name: "David", addr: "321 Oak St", profession: "Artist" },

{ person\_id: 5, name: "Eve", addr: "654 Pine St", profession: "Doctor" },

{ person\_id: 6, name: "Frank", addr: "159 Cedar St", profession: "Engineer" },

{ person\_id: 7, name: "Grace", addr: "753 Birch St", profession: "Artist" }

]);

var mapFunction = function() {

emit(this.profession, this.name); // Emitting profession as the key and name as the value

};

var reduceFunction = function(key, values) {

return values; // Returning the array of names for each profession

};

db.person.mapReduce(

mapFunction,

reduceFunction,

{

out: "profession\_group" // Output collection for the results

}

);

db.profession\_group.find().pretty();

**15) Perform CRUD operation in mongo db –**

**Use : person( person\_id, name, addr, profession )**

**1.Create Collection.**

**2.Inserting data in collection.**

**3.Reading data of collection.**

**4.Updating data of collection.**

**5.Deleting data from collection.**

Ans :

**1. Create Collection**

In MongoDB, collections are created automatically when you first insert data. For this example, let’s assume we’re working within a test database:

use test

**2. Inserting Data in Collection**

// Insert a single document

db.person.insertOne({

person\_id: 1,

name: "John Doe",

addr: "123 Maple St",

profession: "Engineer"

})

// Insert multiple documents

db.person.insertMany([

{

person\_id: 2,

name: "Jane Smith",

addr: "456 Oak St",

profession: "Teacher"

},

{

person\_id: 3,

name: "Robert Brown",

addr: "789 Pine St",

profession: "Doctor"

}

])

**3. Reading Data from Collection**

// Retrieve all documents

db.person.find()

// Retrieve a document with specific fields

db.person.find({ person\_id: 1 })

**4. Updating Data in Collection**

// Update a specific document

db.person.updateOne(

{ person\_id: 1 },

{ $set: { addr: "321 Maple St", profession: "Senior Engineer" } }

)

// Update multiple documents

db.person.updateMany(

{ profession: "Teacher" },

{ $set: { profession: "Educator" } }

)

**5. Deleting Data from Collection**

// Delete a specific document

db.person.deleteOne({ person\_id: 2 })

// Delete multiple documents

db.person.deleteMany({ profession: "Doctor" })

**16) Perform CRUD operation and Aggregation in mongo db**

**employee(emp\_id,e\_name,salary ,Date of Joining,Dapt\_no,Designation)**

**1. Display the count of employee department wise.**

**2. Dsiplay the average salary of employee in sales department.**

**3. Dsiplay minimum salary to employees joins in June 2016**

**4. Display maximum salary given to employee in production department.**

**5. Display record of first and last employee department wise.**

Ans :

db.employee.aggregate([

{

$group: {

\_id: "$Dept\_no", // Group by department number

count: { $sum: 1 } // Count the number of employees in each department

}

}

]);

db.employee.aggregate([

{

$match: {

Designation: "Sales" // Filter for sales department

}

},

{

$group: {

\_id: "$Designation", // Group by designation

averageSalary: { $avg: "$salary" } // Calculate average salary

}

}

]);

db.employee.aggregate([

{

$match: {

Date\_of\_Joining: {

$gte: ISODate("2016-06-01"),

$lt: ISODate("2016-07-01")

}

}

},

{

$group: {

\_id: null, // No grouping, just want the min salary

minimumSalary: { $min: "$salary" } // Find minimum salary

}

}

]);

db.employee.aggregate([

{

$match: {

Designation: "Production" // Filter for production department

}

},

{

$group: {

\_id: "$Designation", // Group by designation

maximumSalary: { $max: "$salary" } // Find maximum salary

}

}

]);

db.employee.aggregate([

{

$sort: { Date\_of\_Joining: 1 } // Sort by date of joining (ascending)

},

{

$group: {

\_id: "$Dept\_no", // Group by department number

firstEmployee: { $first: "$$ROOT" }, // Get the first employee

lastEmployee: { $last: "$$ROOT" } // Get the last employee

}

}

]);

**connectivity**

17) **Consider student ( roll\_no, name ,marks, class) table. Perform add update and delete operation on same table through java program. Write menu driven program.**

Ans :

import java.sql.\*;

import java.util.Scanner;

public class SimpleStudentManager {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/your\_database"; // Update with your database URL

String user = "your\_username"; // Update with your database username

String password = "your\_password"; // Update with your database password

// Establishing connection

Connection connection = DriverManager.getConnection(url, user, password);

Scanner scanner = new Scanner(System.in);

int choice;

// Menu loop

while (true) {

System.out.println("Menu:");

System.out.println("1. Add Student");

System.out.println("2. Update Student");

System.out.println("3. Delete Student");

System.out.println("4. Exit");

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

if (choice == 1) {

// Add Student

System.out.print("Enter roll number: ");

int rollNo = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Enter name: ");

String name = scanner.nextLine();

System.out.print("Enter marks: ");

int marks = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Enter class: ");

String className = scanner.nextLine();

String insertSQL = "INSERT INTO student (roll\_no, name, marks, class) VALUES (?, ?, ?, ?)";

PreparedStatement pstmt = connection.prepareStatement(insertSQL);

pstmt.setInt(1, rollNo);

pstmt.setString(2, name);

pstmt.setInt(3, marks);

pstmt.setString(4, className);

pstmt.executeUpdate();

System.out.println("Student added successfully.");

} else if (choice == 2) {

// Update Student

System.out.print("Enter roll number to update: ");

int rollNo = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Enter new name: ");

String name = scanner.nextLine();

System.out.print("Enter new marks: ");

int marks = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Enter new class: ");

String className = scanner.nextLine();

String updateSQL = "UPDATE student SET name = ?, marks = ?, class = ? WHERE roll\_no = ?";

PreparedStatement pstmt = connection.prepareStatement(updateSQL);

pstmt.setString(1, name);

pstmt.setInt(2, marks);

pstmt.setString(3, className);

pstmt.setInt(4, rollNo);

pstmt.executeUpdate();

System.out.println("Student updated successfully.");

} else if (choice == 3) {

// Delete Student

System.out.print("Enter roll number to delete: ");

int rollNo = scanner.nextInt();

String deleteSQL = "DELETE FROM student WHERE roll\_no = ?";

PreparedStatement pstmt = connection.prepareStatement(deleteSQL);

pstmt.setInt(1, rollNo);

pstmt.executeUpdate();

System.out.println("Student deleted successfully.");

} else if (choice == 4) {

System.out.println("Exiting...");

break;

} else {

System.out.println("Invalid choice. Please try again.");

}

}

// Clean up

scanner.close();

connection.close();

}

}

**Pl/sql**

**18) Implement 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 and900 category 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).**

Ans :

CREATE TABLE Stud\_Marks (

name VARCHAR(255),

total\_marks INT

);

CREATE TABLE Result (

Roll INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(255),

Class VARCHAR(50)

);

DELIMITER $$

CREATE PROCEDURE proc\_Grade()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE student\_name VARCHAR(255);

DECLARE student\_marks INT;

DECLARE student\_cursor CURSOR FOR

SELECT name, total\_marks FROM Stud\_Marks;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN student\_cursor;

read\_loop: LOOP

FETCH student\_cursor INTO student\_name, student\_marks;

IF done THEN

LEAVE read\_loop;

END IF;

-- Determine the class based on total\_marks

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

INSERT INTO Result (Name, Class) VALUES (student\_name, 'Distinction');

ELSEIF student\_marks >= 900 AND student\_marks <= 989 THEN

INSERT INTO Result (Name, Class) VALUES (student\_name, 'First Class');

ELSEIF student\_marks >= 825 AND student\_marks <= 899 THEN

INSERT INTO Result (Name, Class) VALUES (student\_name, 'Higher Second Class');

END IF;

END LOOP;

CLOSE student\_cursor;

END$$

DELIMITER ;

-- Insert sample data into Stud\_Marks

INSERT INTO Stud\_Marks (name, total\_marks) VALUES

('Alice', 1450),

('Bob', 920),

('Charlie', 880),

('David', 1000),

('Eve', 800);

-- Call the stored procedure to categorize students

CALL proc\_Grade();

SELECT \* FROM Result;

**19) Write a database trigger on customer( cust\_id, c\_name, addr) 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 cust\_Audit table.**

Ans :

-- Customer table to hold customer records

CREATE TABLE customer (

cust\_id INT PRIMARY KEY,

c\_name VARCHAR(100),

addr VARCHAR(255)

);

-- Audit table to log changes in customer records

CREATE TABLE cust\_Audit (

audit\_id INT AUTO\_INCREMENT PRIMARY KEY,

action\_type VARCHAR(10),

cust\_id INT,

c\_name VARCHAR(100),

addr VARCHAR(255),

action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER //

CREATE TRIGGER before\_customer\_update

BEFORE UPDATE ON customer

FOR EACH ROW

BEGIN

INSERT INTO cust\_Audit (action\_type, cust\_id, c\_name, addr)

VALUES ('UPDATE', OLD.cust\_id, OLD.c\_name, OLD.addr);

END;

//

DELIMITER ;

DELIMITER //

CREATE TRIGGER before\_customer\_delete

BEFORE DELETE ON customer

FOR EACH ROW

BEGIN

INSERT INTO cust\_Audit (action\_type, cust\_id, c\_name, addr)

VALUES ('DELETE', OLD.cust\_id, OLD.c\_name, OLD.addr);

END;

//

DELIMITER ;

-- Insert sample data into the customer table

INSERT INTO customer (cust\_id, c\_name, addr)

VALUES (1, 'Alice', '123 Maple St'),

(2, 'Bob', '456 Oak St');

-- Update a record to trigger before\_customer\_update

UPDATE customer SET c\_name = 'Alice Johnson' WHERE cust\_id = 1;

-- Delete a record to trigger before\_customer\_delete

DELETE FROM customer WHERE cust\_id = 2;

-- View the cust\_Audit table to check the logs

SELECT \* FROM cust\_Audit;

**20) Implement a database trigger on client\_master( c\_id, c\_name, acc\_no) table. The System should keep track of the records that are being updated or inserted. The old value of updated or deleted records should be added in client\_Audit table.**

Ans :

-- Main client\_master table to hold client information

CREATE TABLE client\_master (

c\_id INT PRIMARY KEY,

c\_name VARCHAR(100),

acc\_no VARCHAR(20)

);

-- Audit table to log changes in client\_master records

CREATE TABLE client\_Audit (

audit\_id INT AUTO\_INCREMENT PRIMARY KEY,

action\_type VARCHAR(10),

c\_id INT,

c\_name VARCHAR(100),

acc\_no VARCHAR(20),

action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER //

CREATE TRIGGER before\_client\_update

BEFORE UPDATE ON client\_master

FOR EACH ROW

BEGIN

INSERT INTO client\_Audit (action\_type, c\_id, c\_name, acc\_no)

VALUES ('UPDATE', OLD.c\_id, OLD.c\_name, OLD.acc\_no);

END;

//

DELIMITER ;

DELIMITER //

CREATE TRIGGER before\_client\_delete

BEFORE DELETE ON client\_master

FOR EACH ROW

BEGIN

INSERT INTO client\_Audit (action\_type, c\_id, c\_name, acc\_no)

VALUES ('DELETE', OLD.c\_id, OLD.c\_name, OLD.acc\_no);

END;

//

DELIMITER ;

DELIMITER //

CREATE TRIGGER after\_client\_insert

AFTER INSERT ON client\_master

FOR EACH ROW

BEGIN

INSERT INTO client\_Audit (action\_type, c\_id, c\_name, acc\_no)

VALUES ('INSERT', NEW.c\_id, NEW.c\_name, NEW.acc\_no);

END;

//

DELIMITER ;

-- Insert sample data into client\_master

INSERT INTO client\_master (c\_id, c\_name, acc\_no)

VALUES (1, 'John Doe', 'ACC123456'),

(2, 'Jane Smith', 'ACC987654');

-- Update a record to trigger before\_client\_update

UPDATE client\_master SET c\_name = 'Johnathan Doe' WHERE c\_id = 1;

-- Delete a record to trigger before\_client\_delete

DELETE FROM client\_master WHERE c\_id = 2;

-- View the client\_Audit table to check the logs

SELECT \* FROM client\_Audit;

**21) Implement a PL/SQL block of code using explicit Cursor, that will merge the data available in the newly created table N\_RollCall 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.**

Ans :

-- Create tables

create table N\_rollcall

(

Roll\_no int ,

name varchar(100)

);

create table O\_rollcall

(

Roll\_no int ,

name varchar(100)

);

-- Insert data into N\_rollcall

insert into N\_rollcall (Roll\_no, name) values

(1, 'john doe'),

(2, 'jane smith'),

(3, 'alice johnson');

-- Insert data into O\_rollcall

insert into O\_rollcall (Roll\_no, name) values

(1, 'john doe'),

(4, 'bob brown');

-- Set delimiter

delimiter //

create procedure update\_rollcall()

begin

-- Declare variables

declare v\_Roll\_no int;

declare v\_name varchar(100);

declare done int default false;

-- Declare cursor

declare cur cursor for

select Roll\_no, name from N\_rollcall;

-- Declare continue handler

declare continue handler for not found set done = true;

-- Open cursor

open cur;

-- Loop through cursor

read\_loop: loop

fetch cur into v\_Roll\_no, v\_name;

if done then

leave read\_loop;

end if;

-- Check if record exists in O\_rollcall

if not exists (select 1 from O\_rollcall where Roll\_no = v\_Roll\_no and name = v\_name) then

insert into O\_rollcall (Roll\_no, name) values (v\_Roll\_no, v\_name);

end if;

end loop;

-- Close cursor

close cur;

end //

DELIMITER ;

CALL update\_rollcall();

select \* from O\_Rollcall;

**22) Write a PL/SQL block of code for the following requirements:- Schema: Borrower(Rollin, Name, DateofIssue, NameofBook, Status) 2. Fine(Roll\_no,Date,Amt) • Accept roll\_no & name of book from user. • Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day. If condition of fine is true, then details will be stored into fine table.**

Ans :

CREATE TABLE Borrower (

Roll\_no INT,

Name VARCHAR(255),

DateofIssue DATE,

NameofBook VARCHAR(255),

Status CHAR(1)

);

CREATE TABLE Fine (

Roll\_no INT,

Date DATE,

Amt INT

);

INSERT INTO Borrower (Roll\_no, Name, DateofIssue, NameofBook, Status)

VALUES (1, 'John Doe', '2023-08-01', 'Book A', 'I'),

(2, 'Jane Smith', '2023-09-01', 'Book B', 'I');

DELIMITER $$

CREATE PROCEDURE CalculateFine (

IN p\_Roll\_no INT,

IN p\_NameofBook VARCHAR(255)

)

BEGIN

DECLARE v\_DateofIssue DATE;

DECLARE v\_Status CHAR(1);

DECLARE v\_Days INT;

DECLARE v\_FineAmt INT DEFAULT 0;

DECLARE v\_Exception EXCEPTION;

-- Get the Date of Issue and Status

SELECT DateofIssue, Status INTO v\_DateofIssue, v\_Status

FROM Borrower

WHERE Roll\_no = p\_Roll\_no AND NameofBook = p\_NameofBook;

-- Calculate the number of days

SET v\_Days = DATEDIFF(CURDATE(), v\_DateofIssue);

-- Calculate the fine amount

IF v\_Days BETWEEN 15 AND 30 THEN

SET v\_FineAmt = v\_Days \* 5;

ELSEIF v\_Days > 30 THEN

SET v\_FineAmt = (30 \* 5) + ((v\_Days - 30) \* 50);

END IF;

-- Update the status to 'R' (Returned)

UPDATE Borrower

SET Status = 'R'

WHERE Roll\_no = p\_Roll\_no AND NameofBook = p\_NameofBook;

-- Insert fine details into Fine table if applicable

IF v\_FineAmt > 0 THEN

INSERT INTO Fine (Roll\_no, Date, Amt)

VALUES (p\_Roll\_no, CURDATE(), v\_FineAmt);

END IF;

-- Exception handling

EXCEPTION

WHEN v\_Exception THEN

-- Handle the exception

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'An error occurred while processing the fine.';

END$$

DELIMITER ;

CALL ProcessBookReturn(123, 'The Great Gatsby');

**23) Implement Basic SQL queries.**

**1. Create table employee.**

**2. Insert 10 records in table.**

**3. Create a view emp\_vl of table employee which has emp\_id , name and dept-attributes.**

**4. Display name and department of employee working in Manager or Marketing department**

**5. Display employees who were hired earliest or latest.**

**6. Display name and department no of employees who are manager, market analysts. Use**

**Predicates**

**List employees hired in August.**

**List employees who are hired after 31/12/2006.**

Ans :

CREATE TABLE employee (

emp\_id INT PRIMARY KEY,

name VARCHAR(100),

dept VARCHAR(50),

hire\_date DATE

);

INSERT INTO employee (emp\_id, name, dept, hire\_date) VALUES

(1, 'Alice', 'Manager', '2020-05-10'),

(2, 'Bob', 'Marketing', '2021-08-15'),

(3, 'Charlie', 'Sales', '2019-03-20'),

(4, 'David', 'Manager', '2018-11-30'),

(5, 'Eva', 'Market Analyst', '2022-01-25'),

(6, 'Frank', 'HR', '2023-02-05'),

(7, 'Grace', 'Marketing', '2020-07-12'),

(8, 'Hank', 'IT', '2016-09-10'),

(9, 'Ivy', 'Manager', '2017-12-01'),

(10, 'Jack', 'Sales', '2019-06-18');

CREATE VIEW emp\_vl AS

SELECT emp\_id, name, dept

FROM employee;

SELECT name, dept

FROM employee

WHERE dept IN ('Manager', 'Marketing');

SELECT \*

FROM employee

WHERE hire\_date = (SELECT MIN(hire\_date) FROM employee);

SELECT \*

FROM employee

WHERE hire\_date = (SELECT MAX(hire\_date) FROM employee);

SELECT name, dept

FROM employee

WHERE dept IN ('Manager', 'Market Analyst');

SELECT \*

FROM employee

WHERE MONTH(hire\_date) = 8;

SELECT \*

FROM employee

WHERE hire\_date > '2006-12-31';

**24) ) Indexing and join: Consider the relation**

**employee (emp\_id,e\_name,salary ,Date of Joining,Dapt\_no,Designation)**

**Customer(c\_id, c\_name , email , city , pincode)**

**Order(order\_id , date , amount , cust\_id.**

**a. create emp\_id as primary key and indices on table employee.**

**b. create user defined index on any column**

**c. create sequence using auto-increment.**

**d. truncate table.**

**e. find list of customers who placed order and details of their orders.**

**f. find info of customers and append order details to the table/**

**g. list down customers who haven’t placed order.**

Ans :

CREATE TABLE employee (

emp\_id INT PRIMARY KEY,

e\_name VARCHAR(100),

salary DECIMAL(10, 2),

date\_of\_joining DATE,

dept\_no INT,

designation VARCHAR(50)

);

-- Create an index on salary

CREATE INDEX idx\_salary ON employee(salary);

CREATE TABLE employee (

emp\_id INT ***AUTO\_INCREMENT*** PRIMARY KEY,

e\_name VARCHAR(100),

salary DECIMAL(10, 2),

date\_of\_joining DATE,

dept\_no INT,

designation VARCHAR(50)

);

TRUNCATE TABLE employee;

SELECT c.c\_id, c.c\_name, o.order\_id, o.date, o.amount

FROM customer c

JOIN order o ON c.c\_id = o.cust\_id;

SELECT c.c\_id, c.c\_name, o.order\_id, o.date, o.amount

FROM customer c

LEFT JOIN order o ON c.c\_id = o.cust\_id;

SELECT c.c\_id, c.c\_name

FROM customer c

LEFT JOIN order o ON c.c\_id = o.cust\_id

WHERE o.order\_id IS NULL;

**25) Implement aggregation and indexing with suitable example in mongodb.**

Ans :

db.students.aggregate([

{

$group: {

\_id: "$class", // Group by the 'class' field

averageMarks: { $avg: "$marks" } // Calculate average marks

}

}

])

db.students.createIndex({ name: 1 }) // Creates an ascending index on the 'name' field