Problem 1: College Admission System
Schema:
● Student(sid INT, name VARCHAR(50), gender VARCHAR(10), dept_id INT)
● Department(dept_id INT, dept_name VARCHAR(50), intake INT)
Questions:
1. Create tables with appropriate keys and constraints.
2. Add 5 students and 3 departments.
3. Display names of all male students and their department names.
4. List departments with more than 2 students using GROUP BY and HAVING.
5. Update the intake to increase by 10% for all departments.

```
-- 1. Create Tables with Appropriate Keys and Constraints
CREATE TABLE Department (
    dept_id INT PRIMARY KEY,
    dept_name VARCHAR(50) NOT NULL,
    intake INT CHECK (intake >= 0)
);

CREATE TABLE Student (
    sid INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    gender VARCHAR(10) CHECK (gender IN ('Male', 'Female')),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);

--  2. Add 5 Students and 3 Departments
-- Inserting into Department
INSERT INTO Department VALUES
(1, 'Computer Science', 60),
(2, 'Mechanical', 50),
(3, 'Electronics', 40);

-- Inserting into Student
INSERT INTO Student VALUES
(101, 'Amit', 'Male', 1),
(102, 'Sneha', 'Female', 2),
(103, 'Ravi', 'Male', 1),
(104, 'Priya', 'Female', 3),
(105, 'Rahul', 'Male', 2);

--  3. Display Names of All Male Students and Their Department Names
SELECT s.name AS Student_Name, d.dept_name AS Department
FROM Student s
JOIN Department d ON s.dept_id = d.dept_id
WHERE s.gender = 'Male';
```

```sql
-- 4. List Departments with More Than 2 Students
SELECT d.dept_name, COUNT(s.sid) AS Student_Count
FROM Student s
JOIN Department d ON s.dept_id = d.dept_id
GROUP BY d.dept_id, d.dept_name
HAVING COUNT(s.sid) > 2;

-- 5. Update the Intake to Increase by 10% for All Departments
SET SQL_SAFE_UPDATES = 0;
UPDATE Department
SET intake = ROUND(intake * 1.10);
SET SQL_SAFE_UPDATES = 1;
```

Problem 2: Online Retail Store
Schema:
● Customers(cust_id INT, name VARCHAR(50), city VARCHAR(30))
● Orders(order_id INT, cust_id INT, amount DECIMAL(10,2), order_date DATE)
Questions:
1. Create both tables with appropriate constraints.
2. Insert at least 4 customers and 5 orders.
3. Display customer names who placed orders above ₹5000.
4. List total order amount placed by each customer in descending order.
5. Retrieve customers who haven’t placed any orders.

```
--  1. Create Both Tables with Appropriate Constraints
CREATE TABLE Customers (
    cust_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    city VARCHAR(30)
);

CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    cust_id INT,
    amount DECIMAL(10,2) CHECK (amount >= 0),
    order_date DATE,
    FOREIGN KEY (cust_id) REFERENCES Customers(cust_id)
);

-- 2. Insert at Least 4 Customers and 5 Orders
-- Inserting Customers
INSERT INTO Customers VALUES
(1, 'Anjali', 'Mumbai'),
(2, 'Raj', 'Delhi'),
(3, 'Sneha', 'Bangalore'),
(4, 'Aman', 'Pune');

-- Inserting Orders
INSERT INTO Orders VALUES
(101, 1, 6500.00, '2024-12-01'),
(102, 2, 4500.00, '2024-12-02'),
(103, 1, 3000.00, '2024-12-03'),
(104, 3, 8000.00, '2024-12-05'),
(105, 2, 10000.00, '2024-12-06');

--  3. Display Customer Names Who Placed Orders Above ₹5000
SELECT DISTINCT c.name
FROM Customers c
JOIN Orders o ON c.cust_id = o.cust_id
WHERE o.amount > 5000;
```

```sql
--  4. List Total Order Amount Placed by Each Customer in Descending Order
SELECT c.name, SUM(o.amount) AS Total_Amount
FROM Customers c
JOIN Orders o ON c.cust_id = o.cust_id
GROUP BY c.cust_id, c.name
ORDER BY Total_Amount DESC;

-- 5. Retrieve Customers Who Haven’t Placed Any Orders
SELECT c.name
FROM Customers c
LEFT JOIN Orders o ON c.cust_id = o.cust_id
WHERE o.order_id IS NULL;
```

Problem 3: Bookstore Inventory
Schema:
● Books(book_id INT, title VARCHAR(100), price DECIMAL(8,2), pub_year INT)
● Sales(sale_id INT, book_id INT, quantity INT, sale_date DATE)
Questions:
1. Create tables with suitable constraints.
2. Insert 4 books and 5 sales records.

3. Display titles of books sold in the year 2024.
4. Show total sales revenue for each book using SUM(price * quantity).
5. Find the title of the most sold book using ORDER BY and LIMIT.

```
CREATE TABLE Books (
    book_id INT PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    price DECIMAL(8,2) CHECK (price >= 0),
    pub_year INT
);

CREATE TABLE Sales (
    sale_id INT PRIMARY KEY,
    book_id INT,
    quantity INT CHECK (quantity > 0),
    sale_date DATE,
    FOREIGN KEY (book_id) REFERENCES Books(book_id)
);

-- Insert Books
INSERT INTO Books VALUES
(1, 'The Alchemist', 350.00, 2015),
(2, 'Atomic Habits', 499.00, 2020),
(3, 'Sapiens', 599.00, 2018),
(4, 'Deep Work', 450.00, 2019);

-- Insert Sales
INSERT INTO Sales VALUES
(101, 1, 3, '2024-01-10'),
(102, 2, 5, '2024-02-15'),
(103, 3, 2, '2023-12-20'),
(104, 1, 4, '2024-03-05'),
(105, 4, 6, '2024-04-01');

-- 3. Display Titles of Books Sold in the Year 2024
SELECT DISTINCT b.title
FROM Books b
JOIN Sales s ON b.book_id = s.book_id
WHERE YEAR(s.sale_date) = 2024;
```

```sql
-- 4. Show Total Sales Revenue for Each Book Using SUM(price * quantity)
SELECT b.title, SUM(b.price * s.quantity) AS total_revenue
FROM Books b
JOIN Sales s ON b.book_id = s.book_id
GROUP BY b.book_id, b.title;

--  5. Find the Title of the Most Sold Book Using ORDER BY and LIMIT
SELECT b.title, SUM(s.quantity) AS total_quantity
FROM Books b
JOIN Sales s ON b.book_id = s.book_id
GROUP BY b.book_id, b.title
ORDER BY total_quantity DESC
LIMIT 1;
```

Problem 4: Airline Reservation
Schema:
● Flights(flight_id INT, source VARCHAR(30), destination VARCHAR(30), fare DECIMAL(6,2))
● Passengers(pid INT, name VARCHAR(50), flight_id INT, travel_date DATE)
Questions:
1. Create both tables with constraints.
2. Insert 3 flights and 5 passenger bookings.
3. List all passengers travelling to ‘Delhi’.
4. Show flight-wise passenger count.
5. Increase fare by 10% for flights having more than 2 bookings.

```
CREATE TABLE Flights (
    flight_id INT PRIMARY KEY,
    source VARCHAR(30),
    destination VARCHAR(30),
    fare DECIMAL(6,2) CHECK (fare >= 0)
);

CREATE TABLE Passengers (
    pid INT PRIMARY KEY,
    name VARCHAR(50),
    flight_id INT,
    travel_date DATE,
    FOREIGN KEY (flight_id) REFERENCES Flights(flight_id)
);

-- 2. Insert 3 Flights and 5 Passenger Bookings
-- Insert Flights
INSERT INTO Flights VALUES
(1, 'Mumbai', 'Delhi', 5000.00),
(2, 'Chennai', 'Bangalore', 3500.00),
(3, 'Kolkata', 'Delhi', 4500.00);

-- Insert Passengers
INSERT INTO Passengers VALUES
(101, 'Aman Singh', 1, '2024-04-22'),
(102, 'Priya Verma', 1, '2024-04-23'),
(103, 'Ravi Kumar', 2, '2024-04-22'),
(104, 'Neha Gupta', 3, '2024-04-23'),
(105, 'Rohit Das', 1, '2024-04-24');

-- 3. List All Passengers Travelling to ‘Delhi’
SELECT p.name, f.destination
FROM Passengers p
JOIN Flights f ON p.flight_id = f.flight_id
WHERE f.destination = 'Delhi';
```

```sql
--  4. Show Flight-Wise Passenger Count
SELECT f.flight_id, f.source, f.destination, COUNT(p.pid) AS passenger_count
FROM Flights f
LEFT JOIN Passengers p ON f.flight_id = p.flight_id
GROUP BY f.flight_id, f.source, f.destination;

--  5. Increase Fare by 10% for Flights Having More Than 2 Bookings
SET SQL_SAFE_UPDATES = 0;
UPDATE Flights
SET fare = fare * 1.10
WHERE flight_id IN (
    SELECT flight_id
    FROM Passengers
    GROUP BY flight_id
    HAVING COUNT(pid) > 2
);
SET SQL_SAFE_UPDATES = 1;
```

Problem 5: Employee Performance Tracker
Schema:
● Employee(emp_id INT, name VARCHAR(50), designation VARCHAR(30), salary INT)
● Performance(emp_id INT, month VARCHAR(15), rating INT)
Questions:
1. Create schema and insert sample data.
2. Find employees with average rating > 4.
3. Display highest rated employee each month.
4. List employees who never received a rating using NOT IN.
5. Display total salary to be paid for ‘Manager’ designation employees.

```
CREATE TABLE Employee (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    designation VARCHAR(30),
    salary INT
);

CREATE TABLE Performance (
    emp_id INT,
    month VARCHAR(15),
    rating INT,
    FOREIGN KEY (emp_id) REFERENCES Employee(emp_id)
);

INSERT INTO Employee VALUES
(1, 'Amit', 'Manager', 80000),
(2, 'Sneha', 'Developer', 60000),
(3, 'Ravi', 'Manager', 85000),
(4, 'Kiran', 'Analyst', 50000),
(5, 'Meena', 'Developer', 62000);

INSERT INTO Performance VALUES
(1, 'January', 5),
(2, 'January', 4),
(3, 'January', 3),
(1, 'February', 5),
(2, 'February', 5),
(4, 'February', 2);

-- Query 1: Employees with Average Rating > 4
SELECT e.emp_id, e.name, AVG(p.rating) AS avg_rating
FROM Employee e
JOIN Performance p ON e.emp_id = p.emp_id
GROUP BY e.emp_id, e.name
HAVING AVG(p.rating) > 4;
```

```sql
-- Query 2: Highest Rated Employee Each Month
SELECT p.month, e.name, p.rating
FROM Performance p
JOIN Employee e ON e.emp_id = p.emp_id
WHERE (p.month, p.rating) IN (
    SELECT month, MAX(rating)
    FROM Performance
    GROUP BY month
);

-- Query 3: Employees Who Never Received a Rating
SELECT name
FROM Employee
WHERE emp_id NOT IN (SELECT emp_id FROM Performance);

-- Query 4: Total Salary of ‘Manager’ Employees
SELECT SUM(salary) AS total_manager_salary
FROM Employee
WHERE designation = 'Manager';
```

Student Performance Tracker
Context:
A university wants to track students' marks across various subjects using MongoDB.
Collection: students
Sample Document:
{
"roll_no": 101,
"name": "Ankita Desai",
"department": "IT",
"marks": [
{ "subject": "DBMS", "score": 78 },
{ "subject": "AI", "score": 89 },
{ "subject": "OS", "score": 91 }
]
}
Tasks:
1. Insert at least 5 student documents with varying subjects and marks.
2. Retrieve all students with more than 85 in "AI".
3. Update the DBMS score of student roll_no: 101 to 85.
4. Delete a student with roll_no: 105.
5. Use aggregation to find the average score in OS across all students.

```
--  Step 1: Create & Use Database
use university

-- Step 2: Create the students Collection (Automatically created on insert)
db.createCollection("students")

-- Step 3: Insert 5 Student Documents
db.students.insertMany([
  {
    roll_no: 101,
    name: "Ankita Desai",
    department: "IT",
    marks: [
      { subject: "DBMS", score: 78 },
      { subject: "AI", score: 89 },
      { subject: "OS", score: 91 }
    ]
  },
  {
    roll_no: 102,
    name: "Rahul Mehta",
    department: "CSE",
    marks: [
      { subject: "DBMS", score: 88 },
      { subject: "AI", score: 92 },
```

```
      { subject: "OS", score: 85 }
    ]
  },
  {
    roll_no: 103,
    name: "Priya Nair",
    department: "ECE",
    marks: [
      { subject: "DBMS", score: 67 },
      { subject: "AI", score: 72 },
      { subject: "OS", score: 79 }
    ]
  },
  {
    roll_no: 104,
    name: "Suresh Kumar",
    department: "IT",
    marks: [
      { subject: "DBMS", score: 91 },
      { subject: "AI", score: 86 },
      { subject: "OS", score: 88 }
    ]
  },
  {
    roll_no: 105,
    name: "Neha Patel",
    department: "CSE",
    marks: [
      { subject: "DBMS", score: 84 },
      { subject: "AI", score: 68 },
      { subject: "OS", score: 80 }
    ]
  }
])

-- Step 4: Find Students with AI > 85
db.students.find({
  marks: {
    $elemMatch: { subject: "AI", score: { $gt: 85 } }
  }
})

-- Step 5: Update DBMS Score of Roll No 101 to 85
db.students.updateOne(
  { roll_no: 101, "marks.subject": "DBMS" },
  { $set: { "marks.$.score": 85 } }
)

-- Step 6: Delete Student with Roll No 105
db.students.deleteOne({ roll_no: 105 })
```

```
-- Step 7: Find Average Score in OS (Across All Students)
db.students.aggregate([
  { $unwind: "$marks" },
  { $match: { "marks.subject": "OS" } },
  {
    $group: {
      _id: null,
      average_OS_score: { $avg: "$marks.score" }
    }
  }
])
```

: Online Bookstore Database
Context:
An online bookstore wants to manage books, authors, and price data.
Collection: books
Sample Document:
{
"title": "The MongoDB Guide",
"author": "Ravi Joshi",

"price": 499,
"category": "Database",
"ratings": [4, 5, 5, 3]
}
Tasks:
1. Insert 5 books with details like title, author, price, category, and rating array.
2. Find all books priced under ₹500.
3. Update the price of a book titled "The MongoDB Guide" to ₹450.
4. Delete all books from category "Old Stock".
5. Use aggregation to calculate the average rating per book.

```
--  Step 1: Create / Use the Database
use bookstore

-- Step 2: Create the books Collection (Automatically created on insert)
db.createCollection("books")

-- Step 3: Insert 5 Books with Details
db.books.insertMany([
  {
    title: "The MongoDB Guide",
    author: "Ravi Joshi",
    price: 499,
    category: "Database",
    ratings: [4, 5, 5, 3]
  },
  {
    title: "Learn Python Programming",
    author: "Alice Williams",
    price: 350,
    category: "Programming",
    ratings: [5, 4, 4, 5]
  },
  {
    title: "Mastering JavaScript",
    author: "Bob Smith",
    price: 650,
```

```
    category: "Programming",
    ratings: [3, 4, 4, 4]
  },
  {
    title: "The Art of Web Design",
    author: "Charles Green",
    price: 750,
    category: "Design",
    ratings: [5, 5, 5, 5]
  },
  {
    title: "Old Stock Book",
    author: "Jane Doe",
    price: 200,
    category: "Old Stock",
    ratings: [2, 3, 2, 1]
  }
])


-- Step 4: Find All Books Priced Under ₹500
db.books.find({ price: { $lt: 500 } })

-- Step 5: Update the Price of a Book Titled "The MongoDB Guide" to ₹450
db.books.updateOne(
  { title: "The MongoDB Guide" },
  { $set: { price: 450 } }
)


--  Step 6: Delete All Books from Category "Old Stock"
db.books.deleteMany({ category: "Old Stock" })


-- Step 7: Use Aggregation to Calculate the Average Rating per Book
db.books.aggregate([
  {
    $project: {
      title: 1,
      avg_rating: { $avg: "$ratings" }
    }
  }
])
```

4: Hospital Patient Records System
Context:
A hospital wants to store and analyze basic patient treatment information.
Collection: patients
Sample Document
{
"patient_id": "P1001",
"name": "Rohan Kulkarni",
"age": 45,
"department": "Cardiology",
"treatments": [
{ "treatment": "ECG", "cost": 1200 },
{ "treatment": "Angiography", "cost": 15000 }
]
}
Tasks:
1. Insert 4–5 patient documents with multiple treatments.
2. Retrieve all patients from "Cardiology".
3. Add a new treatment for patient "P1001".

4. Delete records of patients older than 80 years.
5. Use aggregation to compute the total treatment cost per patient.

```
-- Step 1: Create / Use the Database
use hospital


-- Step 2: Create the patients Collection (Automatically created on insert)
db.createCollection("patients")



--  Step 3: Insert 4–5 Patient Documents with Multiple Treatments
db.patients.insertMany([
  {
    patient_id: "P1001",
    name: "Rohan Kulkarni",
    age: 45,
    department: "Cardiology",
    treatments: [
      { treatment: "ECG", cost: 1200 },
      { treatment: "Angiography", cost: 15000 }
    ]
  },
  {
    patient_id: "P1002",
```

```
    name: "Suman Reddy",
    age: 60,
    department: "Orthopedics",
    treatments: [
      { treatment: "X-ray", cost: 800 },
      { treatment: "Hip Replacement", cost: 30000 }
    ]
  },
  {
    patient_id: "P1003",
    name: "Ajay Sharma",
    age: 35,
    department: "Cardiology",
    treatments: [
      { treatment: "ECG", cost: 1200 },
      { treatment: "Stress Test", cost: 5000 }
    ]
  },
  {
    patient_id: "P1004",
    name: "Priya Singh",
    age: 55,
    department: "Neurology",
    treatments: [
      { treatment: "MRI Scan", cost: 4000 },
      { treatment: "EEG", cost: 2500 }
    ]
  },
  {
    patient_id: "P1005",
    name: "Vikram Gupta",
    age: 85,
    department: "Cardiology",
    treatments: [
      { treatment: "ECG", cost: 1200 },
      { treatment: "Angioplasty", cost: 25000 }
    ]
  }
])


-- Step 4: Retrieve All Patients from "Cardiology"
db.patients.find({ department: "Cardiology" })


-- Step 5: Add a New Treatment for Patient "P1001"
db.patients.updateOne(
  { patient_id: "P1001" },
  { $push: { treatments: { treatment: "Heart Surgery", cost: 50000 } } }
)
```

```
-- Step 6: Delete Records of Patients Older Than 80 Years
db.patients.deleteMany({ age: { $gt: 80 } })


--  Step 7: Use Aggregation to Compute the Total Treatment Cost per Patient
db.patients.aggregate([
  {
    $project: {
      patient_id: 1,
      total_treatment_cost: { $sum: "$treatments.cost" }
    }
  }
])
```

Problem 4: Movie Ratings and Reviews
Context:
A movie platform stores user reviews and wants to perform analysis on the data.
Collection: movies
Sample Document:
```
{
"movie_id": 1,
"title": "Interstellar",
"genre": "Sci-Fi",
"release_year": 2014,
"ratings": [
{ "user": "user1", "score": 5 },
{ "user": "user2", "score": 4 }
]
}
```
Tasks:
1. Insert at least 5 movie documents with ratings.
2. Find all movies released after 2010 in the "Sci-Fi" genre.
3. Update the title of a movie from "Inception" to "Inception (2010)".
4. Delete all movies with an average rating below 3.
5. Use aggregation to calculate the average score of each movie.

```
-- Step 1: Create / Use the Database
use movie_platform


--Step 2: Create the movies Collection (Automatically created on insert)
db.createCollection("movies")


--  Step 3: Insert at Least 5 Movie Documents with Ratings
db.movies.insertMany([
  {
    movie_id: 1,
    title: "Interstellar",
    genre: "Sci-Fi",
    release_year: 2014,
    ratings: [
      { user: "user1", score: 5 },
      { user: "user2", score: 4 }
    ]
  },
  {
    movie_id: 2,
    title: "Inception",
    genre: "Sci-Fi",
```

```
    release_year: 2010,
    ratings: [
      { user: "user1", score: 5 },
      { user: "user2", score: 3 }
    ]
  },
  {
    movie_id: 3,
    title: "The Dark Knight",
    genre: "Action",
    release_year: 2008,
    ratings: [
      { user: "user1", score: 5 },
      { user: "user3", score: 4 }
    ]
  },
  {
    movie_id: 4,
    title: "The Matrix",
    genre: "Sci-Fi",
    release_year: 1999,
    ratings: [
      { user: "user2", score: 4 },
      { user: "user3", score: 4 }
    ]
  },
  {
    movie_id: 5,
    title: "Avatar",
    genre: "Sci-Fi",
    release_year: 2009,
    ratings: [
      { user: "user1", score: 3 },
      { user: "user2", score: 2 }
    ]
  }
])


--  Step 4: Find All Movies Released After 2010 in the "Sci-Fi" Genre
db.movies.find({
  release_year: { $gt: 2010 },
  genre: "Sci-Fi"
})


-- Step 5: Update the Title of a Movie from "Inception" to "Inception (2010)"
db.movies.updateOne(
  { title: "Inception" },
  { $set: { title: "Inception (2010)" } }
```

```
)


-- Step 6: Delete All Movies with an Average Rating Below 3
db.movies.deleteMany({
  $expr: {
    $lt: [
      { $avg: "$ratings.score" },
      3
    ]
  }
})



-- Step 7: Use Aggregation to Calculate the Average Score of Each Movie
db.movies.aggregate([
  {
    $project: {
      movie_id: 1,
      title: 1,
      average_rating: { $avg: "$ratings.score" }
    }
  }
])
```

A company wants to give a bonus of ₹5000 to employees whose salaries are less than ₹30,000. The
HR department maintains a database of employee records.
Schema:
● Employees(emp_id INT PRIMARY KEY, name VARCHAR(50), salary INT, bonus INT DEFAULT 0)

Tasks:
1.Write a stored procedure using a cursor that:
● Retrieves all employees with salary < ₹30,000
● Adds ₹5000 to their bonus column
● Displays their name and updated bonus value

```
-- Stored Procedure: give_bonus_to_low_salary_employees
DELIMITER $$

CREATE PROCEDURE give_bonus_to_low_salary_employees()
BEGIN
    -- Declare variables
    DECLARE done INT DEFAULT FALSE;
    DECLARE empName VARCHAR(50);
    DECLARE empId INT;
    DECLARE currentBonus INT;

    -- Cursor to select employees with salary < 30000
    DECLARE emp_cursor CURSOR FOR
        SELECT emp_id, name, bonus FROM Employees WHERE salary < 30000;

    -- Handler to exit loop
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Open cursor
    OPEN emp_cursor;

    read_loop: LOOP
        FETCH emp_cursor INTO empId, empName, currentBonus;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Update bonus
        UPDATE Employees
        SET bonus = bonus + 5000
        WHERE emp_id = empId;

        -- Show employee name and updated bonus
        SELECT empName AS Name, currentBonus + 5000 AS UpdatedBonus;
```

```
    END LOOP;

    -- Close cursor
    CLOSE emp_cursor;
END $$

DELIMITER ;

--  To Execute the Procedure:
CALL give_bonus_to_low_salary_employees();

-- Example Table Setup (if needed):
CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    salary INT,
    bonus INT DEFAULT 0
);

INSERT INTO Employees (emp_id, name, salary)
VALUES
(1, 'Amit', 25000),
(2, 'Neha', 32000),
(3, 'Ravi', 28000),
(4, 'Pooja', 35000);
```

2. A library tracks borrowed books and their return status. A fine of ₹2 is applied for each day after
the due date.
Schema:
● Borrowers(borrow_id INT PRIMARY KEY, student_name VARCHAR(50), due_date DATE, return_date DATE, fine INT DEFAULT 0)
Task:
Write a stored procedure using a cursor to:
● Loop through all records in Borrowers
● For each student who returned the book late, calculate the number of overdue days
● Multiply overdue days by ₹2 and update the fine column
● Show a message like: Fine of ₹20 updated for Rahul Singh

```
-- Stored Procedure: calculate_fines
DELIMITER $$

CREATE PROCEDURE calculate_fines()
BEGIN
    -- Declare variables
    DECLARE done INT DEFAULT FALSE;
    DECLARE borrowId INT;
    DECLARE studentName VARCHAR(50);
    DECLARE dueDate DATE;
    DECLARE returnDate DATE;
    DECLARE overdueDays INT;
    DECLARE fineAmount INT;

    -- Declare cursor for Borrowers table
    DECLARE borrower_cursor CURSOR FOR
        SELECT borrow_id, student_name, due_date, return_date
        FROM Borrowers;

    -- Handler to exit the loop
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Open the cursor
    OPEN borrower_cursor;

    read_loop: LOOP
        FETCH borrower_cursor INTO borrowId, studentName, dueDate, returnDate;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Check if the book was returned late
        IF returnDate > dueDate THEN
            SET overdueDays = DATEDIFF(returnDate, dueDate);
```

```
            SET fineAmount = overdueDays * 2;

            -- Update fine in the table
            UPDATE Borrowers
            SET fine = fineAmount
            WHERE borrow_id = borrowId;

            -- Show message
            SELECT CONCAT('Fine of ₹', fineAmount, ' updated for ', studentName) AS
Message;
        END IF;
    END LOOP;

    -- Close the cursor
    CLOSE borrower_cursor;
END$$

DELIMITER ;

-- Sample Table Setup (if needed):
CREATE TABLE Borrowers (
    borrow_id INT PRIMARY KEY,
    student_name VARCHAR(50),
    due_date DATE,
    return_date DATE,
    fine INT DEFAULT 0
);

INSERT INTO Borrowers (borrow_id, student_name, due_date, return_date)
VALUES
(1, 'Rahul Singh', '2024-04-10', '2024-04-15'),
(2, 'Anjali Mehta', '2024-04-12', '2024-04-12'),
(3, 'Vikram Rao', '2024-04-05', '2024-04-08');

-- To Execute the Procedure:
CALL calculate_fines();
```

1: Track Salary Updates
Context:
A company wants to maintain a log of all salary changes for employees. Every time an employee’s
salary is updated, the old and new values should be stored in a separate table for audit purposes.
Tables:
● employees(emp_id INT PRIMARY KEY, name VARCHAR(50), salary DECIMAL(10,2))
● salary_log(log_id INT AUTO_INCREMENT PRIMARY KEY, emp_id INT, old_salary DECIMAL(10,2), new_salary DECIMAL(10,2), change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP)
Objective:
Create a BEFORE UPDATE trigger on the employees table that:
● Captures the old and new salary values whenever salary is updated
● Inserts them into the salary_log table

```
-- Step 1: Create Tables
-- Employees table
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    salary DECIMAL(10,2)
);

-- Salary log table
CREATE TABLE salary_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    emp_id INT,
    old_salary DECIMAL(10,2),
    new_salary DECIMAL(10,2),
    change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Step 2: Create Trigger
DELIMITER $$

CREATE TRIGGER before_salary_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
    -- Only log if salary actually changes
    IF OLD.salary != NEW.salary THEN
        INSERT INTO salary_log (emp_id, old_salary, new_salary)
        VALUES (OLD.emp_id, OLD.salary, NEW.salary);
    END IF;
END$$
```

```
DELIMITER ;

-- Example Usage
-- Insert an employee
INSERT INTO employees (emp_id, name, salary)
VALUES (1, 'Anita Sharma', 30000.00);

-- Update salary (this will trigger the log)
UPDATE employees
SET salary = 35000.00
WHERE emp_id = 1;

-- View the log
SELECT * FROM salary_log;
```