**Procedure**

-- Step 1: Create Database

CREATE DATABASE IF NOT EXISTS CompanyDB;

USE CompanyDB;

-- Step 2: Create Employees Table

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(50),

salary INT,

bonus INT DEFAULT 0

);

-- Step 3: Insert Sample Employees

INSERT INTO Employees (emp\_id, name, salary) VALUES

(1, 'Ravi Kumar', 25000),

(2, 'Sneha Patel', 32000),

(3, 'Aman Verma', 28000),

(4, 'Megha Shah', 45000),

(5, 'Rajesh Yadav', 22000);

-- Step 4: Create Stored Procedure using Cursor

DELIMITER $$

CREATE PROCEDURE UpdateBonus()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE empName VARCHAR(50);

DECLARE empID INT;

DECLARE empBonus INT;

DECLARE emp\_cursor CURSOR FOR

SELECT emp\_id, name FROM Employees WHERE salary < 30000;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN emp\_cursor;

read\_loop: LOOP

FETCH emp\_cursor INTO empID, empName;

IF done THEN

LEAVE read\_loop;

END IF;

UPDATE Employees SET bonus = bonus + 5000 WHERE emp\_id = empID;

SELECT bonus INTO empBonus FROM Employees WHERE emp\_id = empID;

SELECT empName AS Employee\_Name, empBonus AS Updated\_Bonus;

END LOOP;

CLOSE emp\_cursor;

END $$

DELIMITER ;

-- Step 5: Call the Procedure to Apply Bonus

CALL UpdateBonus();

**2.Library**

-- Step 1: Create Database

CREATE DATABASE IF NOT EXISTS LibraryDB;

USE LibraryDB;

-- Step 2: Create Borrowers Table

CREATE TABLE Borrowers (

borrow\_id INT PRIMARY KEY,

student\_name VARCHAR(50),

due\_date DATE,

return\_date DATE,

fine INT DEFAULT 0

);

-- Step 3: Insert Sample Data

INSERT INTO Borrowers (borrow\_id, student\_name, due\_date, return\_date) VALUES

(1, 'Rahul Singh', '2025-04-10', '2025-04-15'),

(2, 'Priya Sharma', '2025-04-12', '2025-04-12'),

(3, 'Ankit Mehra', '2025-04-08', '2025-04-18'),

(4, 'Neha Joshi', '2025-04-14', '2025-04-14'),

(5, 'Karan Yadav', '2025-04-10', '2025-04-11');

-- Step 4: Create Stored Procedure to Calculate and Update Fines

DELIMITER $$

CREATE PROCEDURE UpdateFines()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE b\_id INT;

DECLARE s\_name VARCHAR(50);

DECLARE d\_date DATE;

DECLARE r\_date DATE;

DECLARE days\_late INT;

DECLARE fine\_amt INT;

DECLARE borrow\_cursor CURSOR FOR

SELECT borrow\_id, student\_name, due\_date, return\_date FROM Borrowers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN borrow\_cursor;

read\_loop: LOOP

FETCH borrow\_cursor INTO b\_id, s\_name, d\_date, r\_date;

IF done THEN

LEAVE read\_loop;

END IF;

-- Calculate overdue days

IF r\_date > d\_date THEN

SET days\_late = DATEDIFF(r\_date, d\_date);

SET fine\_amt = days\_late \* 2;

-- Update the fine

UPDATE Borrowers SET fine = fine\_amt WHERE borrow\_id = b\_id;

-- Show message

SELECT CONCAT('Fine of ₹', fine\_amt, ' updated for ', s\_name) AS Message;

END IF;

END LOOP;

CLOSE borrow\_cursor;

END $$

DELIMITER ;

-- Step 5: Call the Procedure

CALL UpdateFines();

**Trigger**

-- Step 1: Create Database

CREATE DATABASE IF NOT EXISTS CompanyAuditDB;

USE CompanyAuditDB;

-- Step 2: Create Employees Table

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(50),

salary DECIMAL(10,2)

);

-- Step 3: Create 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 4: Insert Sample Data

INSERT INTO employees (emp\_id, name, salary) VALUES

(1, 'Amit Sharma', 50000.00),

(2, 'Priya Verma', 62000.00),

(3, 'Kunal Mehta', 45000.00);

-- Step 5: Create BEFORE UPDATE Trigger

DELIMITER $$

CREATE TRIGGER before\_salary\_update

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

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 ;

-- Step 6: Example Salary Update (Try this to test trigger)

UPDATE employees SET salary = 53000.00 WHERE emp\_id = 1;

-- Step 7: Check the log table

SELECT \* FROM salary\_log;

**Mongodb**

Student performance

// Step 1: Create or switch to the database

use StudentPerformanceDB;

// Step 2: Drop collection if already exists (for fresh execution)

db.students.drop();

// Step 3: Insert 5 students

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: 72 },

{ subject: "AI", score: 86 },

{ subject: "OS", score: 67 }

]

},

{

roll\_no: 103,

name: "Sneha Patil",

department: "IT",

marks: [

{ subject: "DBMS", score: 88 },

{ subject: "AI", score: 76 },

{ subject: "OS", score: 94 }

]

},

{

roll\_no: 104,

name: "Vikram Rao",

department: "ECE",

marks: [

{ subject: "DBMS", score: 81 },

{ subject: "AI", score: 90 },

{ subject: "OS", score: 85 }

]

},

{

roll\_no: 105,

name: "Neha Joshi",

department: "CSE",

marks: [

{ subject: "DBMS", score: 65 },

{ subject: "AI", score: 70 },

{ subject: "OS", score: 73 }

]

}

]);

print("\n Inserted 5 student documents.\n");

// Step 4: Find students with AI score > 85

print(" Students with AI score > 85:");

db.students.find({

marks: {

$elemMatch: {

subject: "AI",

score: { $gt: 85 }

}

}

}).pretty();

// Step 5: Update DBMS score of student with roll\_no 101 to 85

db.students.updateOne(

{ roll\_no: 101, "marks.subject": "DBMS" },

{ $set: { "marks.$.score": 85 } }

);

print("\n Updated DBMS score to 85 for roll\_no 101");

// Step 6: Delete student with roll\_no 105

db.students.deleteOne({ roll\_no: 105 });

print("\n🗑️ Deleted student with roll\_no 105");

// Step 7: Aggregation - Average OS score

print("\n Average OS score across all students:");

db.students.aggregate([

{ $unwind: "$marks" },

{ $match: { "marks.subject": "OS" } },

{

$group: {

\_id: null,

average\_OS\_score: { $avg: "$marks.score" }

}

}

]).forEach(print);

Bookstore

// 1. Select or create the database

use OnlineBookstoreDB;

// 2. Drop existing 'books' collection if re-running the script

db.books.drop();

// 3. Insert 5 book documents

db.books.insertMany([

{

title: "The MongoDB Guide",

author: "Ravi Joshi",

price: 499,

category: "Database",

ratings: [4, 5, 5, 3]

},

{

title: "Learning Python",

author: "Alka Verma",

price: 650,

category: "Programming",

ratings: [5, 5, 4, 4]

},

{

title: "HTML & CSS Basics",

author: "Rohit Sharma",

price: 350,

category: "Web Design",

ratings: [3, 4, 4]

},

{

title: "Data Structures in Java",

author: "Priya Nair",

price: 475,

category: "Programming",

ratings: [5, 5, 4, 5]

},

{

title: "Old Tales Collection",

author: "Unknown",

price: 299,

category: "Old Stock",

ratings: [2, 3]

}

]);

print("\nInserted 5 book documents.\n");

// 4. Find all books priced under ₹500

print("Books priced under ₹500:");

db.books.find({ price: { $lt: 500 } }).pretty();

// 5. Update the price of "The MongoDB Guide" to ₹450

db.books.updateOne(

{ title: "The MongoDB Guide" },

{ $set: { price: 450 } }

);

print("\nUpdated price of 'The MongoDB Guide' to ₹450");

// 6. Delete all books from category "Old Stock"

db.books.deleteMany({ category: "Old Stock" });

print("\nDeleted all books from category 'Old Stock'");

// 7. Aggregation: Average rating per book

print("\nAverage rating per book:");

db.books.aggregate([

{

$project: {

title: 1,

avg\_rating: { $avg: "$ratings" }

}

}

]).forEach(print);

Hospital

// 1. Select or create the database

use HospitalDB;

// 2. Drop existing 'patients' collection if re-running the script

db.patients.drop();

// 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: "Priya Singh",

age: 30,

department: "Orthopedics",

treatments: [

{ treatment: "X-ray", cost: 800 },

{ treatment: "Physiotherapy", cost: 2500 }

]

},

{

patient\_id: "P1003",

name: "Amit Verma",

age: 70,

department: "Neurology",

treatments: [

{ treatment: "MRI", cost: 5000 },

{ treatment: "CT Scan", cost: 3500 }

]

},

{

patient\_id: "P1004",

name: "Neha Joshi",

age: 55,

department: "Cardiology",

treatments: [

{ treatment: "ECG", cost: 1200 },

{ treatment: "Angioplasty", cost: 12000 }

]

},

{

patient\_id: "P1005",

name: "Suresh Patel",

age: 85,

department: "Geriatrics",

treatments: [

{ treatment: "Blood Test", cost: 400 },

{ treatment: "X-ray", cost: 800 }

]

}

]);

print("\nInserted 5 patient documents.\n");

// 4. Retrieve all patients from "Cardiology"

print("Patients in Cardiology department:");

db.patients.find({ department: "Cardiology" }).pretty();

// 5. Add a new treatment for patient "P1001"

db.patients.updateOne(

{ patient\_id: "P1001" },

{ $push: { treatments: { treatment: "Stress Test", cost: 5000 } } }

);

print("\nAdded new treatment for P1001: Stress Test with cost ₹5000");

// 6. Delete records of patients older than 80 years

db.patients.deleteMany({ age: { $gt: 80 } });

print("\nDeleted records of patients older than 80 years");

// 7. Aggregation to compute the total treatment cost per patient

print("\nTotal treatment cost per patient:");

db.patients.aggregate([

{

$project: {

name: 1,

total\_cost: { $sum: "$treatments.cost" }

}

}

]).forEach(print);

Movie

// 1. Select or create the database

use MoviePlatformDB;

// 2. Drop existing 'movies' collection if re-running the script

db.movies.drop();

// 3. Insert 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: "user3", score: 5 },

{ user: "user4", score: 4 }

]

},

{

movie\_id: 3,

title: "The Dark Knight",

genre: "Action",

release\_year: 2008,

ratings: [

{ user: "user5", score: 5 },

{ user: "user6", score: 4 }

]

},

{

movie\_id: 4,

title: "The Matrix",

genre: "Sci-Fi",

release\_year: 1999,

ratings: [

{ user: "user7", score: 5 },

{ user: "user8", score: 3 }

]

},

{

movie\_id: 5,

title: "The Prestige",

genre: "Drama",

release\_year: 2006,

ratings: [

{ user: "user9", score: 4 },

{ user: "user10", score: 2 }

]

}

]);

print("\nInserted 5 movie documents.\n");

// 4. Find all movies released after 2010 in the "Sci-Fi" genre

print("Movies released after 2010 in the 'Sci-Fi' genre:");

db.movies.find({

release\_year: { $gt: 2010 },

genre: "Sci-Fi"

}).pretty();

// 5. Update the title of a movie from "Inception" to "Inception (2010)"

db.movies.updateOne(

{ title: "Inception" },

{ $set: { title: "Inception (2010)" } }

);

print("\nUpdated title of 'Inception' to 'Inception (2010)'");

// 6. Delete all movies with an average rating below 3

db.movies.deleteMany({

$expr: {

$lt: [

{ $avg: "$ratings.score" },

3

]

}

});

print("\nDeleted movies with an average rating below 3");

// 7. Aggregation to calculate the average score of each movie

print("\nAverage score of each movie:");

db.movies.aggregate([

{

$project: {

title: 1,

avg\_rating: { $avg: "$ratings.score" }

}

}

]).forEach(print);