-- SUPPLIER table

CREATE TABLE SUPPLIER (

Sno VARCHAR(5) PRIMARY KEY CHECK (Sno LIKE 'S%' AND TRY\_CAST(SUBSTRING(Sno, 2, LEN(Sno)) AS INT) BETWEEN 0 AND 9999),

Sname VARCHAR(50) NOT NULL,

address VARCHAR(100) NOT NULL,

City VARCHAR(50) NOT NULL CHECK (City IN ('London', 'Paris', 'Rome', 'New York', 'Amsterdam'))

);

-- PARTS table

CREATE TABLE PARTS (

Pno INT PRIMARY KEY,

Pname VARCHAR(50) NOT NULL,

Color VARCHAR(20) NOT NULL,

Weight FLOAT NOT NULL,

Price DECIMAL(10,2) NOT NULL

);

-- PROJECT table

CREATE TABLE PROJECT (

Jno INT PRIMARY KEY,

Jname VARCHAR(50) NOT NULL UNIQUE,

City VARCHAR(50) NOT NULL CHECK (City IN ('London', 'Paris', 'Rome', 'New York', 'Amsterdam'))

);

-- SPJ table (Supply information)

CREATE TABLE SPJ (

Sno VARCHAR(5) NOT NULL,

Pno INT NOT NULL,

Jno INT NOT NULL,

Qty INT NOT NULL,

FOREIGN KEY (Sno) REFERENCES SUPPLIER(Sno),

FOREIGN KEY (Pno) REFERENCES PARTS(Pno),

FOREIGN KEY (Jno) REFERENCES PROJECT(Jno)

);

-- SUPPLIER

INSERT INTO SUPPLIER VALUES

('S001', 'John Supplies', '221B Baker St', 'London'),

('S002', 'Paris Goods', '10 Rue St', 'Paris'),

('S003', 'Rome Deals', '5 Roma Ave', 'Rome'),

('S004', 'NY Inc.', 'Wall Street', 'New York'),

('S005', 'AMS Traders', 'Tulip Blvd', 'Amsterdam'),

('S006', 'Speedy Supply', 'King's Road', 'London'),

('S007', 'EuroSupply', 'Roma Central', 'Rome'),

('S008', 'WestCo', '5th Avenue', 'New York'),

('S009', 'NorthHub', 'North Square', 'London'),

('S010', 'Global Parts', 'Trade Street', 'Amsterdam');

-- PARTS

INSERT INTO PARTS VALUES

(1, 'Bolt', 'Red', 0.5, 2.00),

(2, 'Nut', 'Blue', 0.3, 1.50),

(3, 'Screw', 'Green', 0.2, 0.75),

(4, 'Plate', 'Silver', 1.0, 5.00),

(5, 'Wheel', 'Black', 2.5, 15.00),

(6, 'Pipe', 'Grey', 3.0, 10.00),

(7, 'Gear', 'White', 1.2, 7.50),

(8, 'Rod', 'Yellow', 2.0, 6.00),

(9, 'Spring', 'Blue', 0.8, 3.25),

(10, 'Clamp', 'Red', 0.4, 2.50);

-- PROJECT

INSERT INTO PROJECT VALUES

(101, 'MetroBuild', 'London'),

(102, 'SkyBridge', 'Paris'),

(103, 'ColosseumRevamp', 'Rome'),

(104, 'NYHighway', 'New York'),

(105, 'CanalWorks', 'Amsterdam'),

(106, 'EcoCity', 'London'),

(107, 'TechPark', 'Paris'),

(108, 'CityLights', 'Rome'),

(109, 'GreenZone', 'Amsterdam'),

(110, 'BridgeX', 'Paris');

-- SPJ (Supply Info)

INSERT INTO SPJ VALUES

('S001', 1, 101, 10),

('S002', 2, 102, 20),

('S003', 3, 103, 15),

('S004', 4, 104, 12),

('S005', 5, 105, 8),

('S006', 6, 106, 9),

('S007', 7, 107, 6),

('S008', 8, 108, 11),

('S009', 9, 109, 10),

('S010', 10, 110, 5),

('S001', 2, 101, 8),

('S002', 3, 102, 6),

('S002', 4, 102, 4); -- So project 102 gets 3 parts

**a) Find all projects which are provided 3 or more parts**

SELECT Jno, COUNT(DISTINCT Pno) AS PartCount

FROM SPJ

GROUP BY Jno

HAVING COUNT(DISTINCT Pno) >= 3;

**b)Trigger to prevent duplicate Jname on insert/update**

CREATE TRIGGER trg\_unique\_jname

ON PROJECT

INSTEAD OF INSERT, UPDATE

AS

BEGIN

IF EXISTS (

SELECT Jname

FROM inserted

WHERE Jname IN (SELECT Jname FROM PROJECT)

)

BEGIN

RAISERROR('Project name must be unique.', 16, 1);

RETURN;

END

ELSE

BEGIN

IF EXISTS (SELECT \* FROM inserted)

BEGIN

INSERT INTO PROJECT (Jno, Jname, City)

SELECT Jno, Jname, City FROM inserted;

END

END

END;

**c) Find full details of all projects in London**

SELECT \* FROM PROJECT

WHERE City = 'London';

**d) Procedure: Total sales of parts supplied to Paris projects**

CREATE PROCEDURE GetTotalSalesForParis

AS

BEGIN

SELECT

P.Pno,

P.Pname,

SUM(S.Qty \* P.Price) AS TotalSales

FROM SPJ S

JOIN PROJECT J ON S.Jno = J.Jno

JOIN PARTS P ON S.Pno = P.Pno

WHERE J.City = 'Paris'

GROUP BY P.Pno, P.Pname;

END;

-- To run:

EXEC GetTotalSalesForParis;

-- PRODUCT table

CREATE TABLE PRODUCT (

Maker VARCHAR(20) NOT NULL CHECK (Maker IN ('IBM', 'Compaq', 'Dell', 'HP', 'Lenovo')),

Modelno INT PRIMARY KEY,

Type VARCHAR(10) NOT NULL CHECK (Type IN ('PC', 'Laptop', 'Printer'))

);

-- PC table

CREATE TABLE PC (

Modelno INT PRIMARY KEY,

Speed INT NOT NULL, -- MHz

RAM INT NOT NULL, -- MB

HD INT NOT NULL, -- GB

CD VARCHAR(10) NOT NULL, -- e.g., '24x'

Price DECIMAL(10,2) NOT NULL,

FOREIGN KEY (Modelno) REFERENCES PRODUCT(Modelno)

);

-- LAPTOP table

CREATE TABLE LAPTOP (

Modelno INT PRIMARY KEY,

Speed INT NOT NULL,

RAM INT NOT NULL,

HD INT NOT NULL,

Price DECIMAL(10,2) NOT NULL,

FOREIGN KEY (Modelno) REFERENCES PRODUCT(Modelno)

);

-- PRINTER table

CREATE TABLE PRINTER (

Modelno INT PRIMARY KEY,

Color CHAR(1) NOT NULL CHECK (Color IN ('T', 'F')),

Type VARCHAR(15) NOT NULL CHECK (Type IN ('laser', 'ink-jet', 'dot-matrix', 'dry')),

Price DECIMAL(10,2) NOT NULL,

FOREIGN KEY (Modelno) REFERENCES PRODUCT(Modelno)

);

-- PRODUCT data

INSERT INTO PRODUCT VALUES

('IBM', 101, 'PC'),

('IBM', 102, 'Laptop'),

('Compaq', 103, 'PC'),

('HP', 104, 'Printer'),

('Dell', 105, 'Laptop'),

('Lenovo', 106, 'PC'),

('HP', 107, 'Laptop'),

('IBM', 108, 'Printer'),

('Compaq', 109, 'Laptop'),

('Dell', 110, 'Printer');

-- PC data

INSERT INTO PC VALUES

(101, 160, 2048, 500, '24x', 32000),

(103, 200, 4096, 1000, '32x', 45000),

(106, 180, 2048, 750, '16x', 40000);

-- LAPTOP data

INSERT INTO LAPTOP VALUES

(102, 220, 8192, 512, 52000),

(105, 250, 16384, 1000, 75000),

(107, 200, 4096, 512, 35000),

(109, 180, 4096, 256, 30000);

-- PRINTER data

INSERT INTO PRINTER VALUES

(104, 'F', 'laser', 8000),

(108, 'T', 'ink-jet', 12000),

(110, 'F', 'dot-matrix', 5000);

**a) Find PC models having a speed of at least 150 MHz**

SELECT \* FROM PC

WHERE Speed >= 150;

**b) Find manufacturers that sell laptops but not PCs**

SELECT DISTINCT P1.Maker

FROM PRODUCT P1

WHERE P1.Type = 'Laptop'

AND NOT EXISTS (

SELECT 1 FROM PRODUCT P2

WHERE P2.Maker = P1.Maker AND P2.Type = 'PC'

);

**c) Trigger on LAPTOP table: price must not be less than 30000**

CREATE TRIGGER trg\_check\_laptop\_price

ON LAPTOP

INSTEAD OF INSERT

AS

BEGIN

IF EXISTS (SELECT \* FROM inserted WHERE Price < 30000)

BEGIN

RAISERROR('Laptop price must not be less than 30000.', 16, 1);

END

ELSE

BEGIN

INSERT INTO LAPTOP (Modelno, Speed, RAM, HD, Price)

SELECT Modelno, Speed, RAM, HD, Price FROM inserted;

END

END;

**d) Procedure to find the manufacturer of the most expensive laptop**

CREATE PROCEDURE GetMaxPricedLaptopMaker

AS

BEGIN

SELECT TOP 1 P.Maker, L.Modelno, L.Price

FROM LAPTOP L

JOIN PRODUCT P ON L.Modelno = P.Modelno

ORDER BY L.Price DESC;

END;

-- To run:

EXEC GetMaxPricedLaptopMaker;

**a) Find the different types of printers produced by Epson**

SELECT DISTINCT PR.Type

FROM PRODUCT P

JOIN PRINTER PR ON P.Modelno = PR.Modelno

WHERE P.Maker = 'Epson';

**b) Find hard disk sizes that occur in two or more PCs**

SELECT HD, COUNT(\*) AS CountOfPCs

FROM PC

GROUP BY HD

HAVING COUNT(\*) >= 2;

**c) Trigger on LAPTOP: Speed must be at least 1200 MHz**

CREATE TRIGGER trg\_min\_laptop\_speed

ON LAPTOP

INSTEAD OF INSERT

AS

BEGIN

IF EXISTS (SELECT \* FROM inserted WHERE Speed < 1200)

BEGIN

RAISERROR('Laptop speed must be at least 1200 MHz.', 16, 1);

RETURN;

END

ELSE

BEGIN

INSERT INTO LAPTOP (Modelno, Speed, RAM, HD, Price)

SELECT Modelno, Speed, RAM, HD, Price FROM inserted;

END

END;

**d) Demonstrate Cursor Using PRODUCT Table**

DECLARE @Maker VARCHAR(20), @Modelno INT, @Type VARCHAR(10);

DECLARE cur\_product CURSOR FOR

SELECT Maker, Modelno, Type FROM PRODUCT;

OPEN cur\_product;

FETCH NEXT FROM cur\_product INTO @Maker, @Modelno, @Type;

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'Maker: ' + @Maker + ', Model No: ' + CAST(@Modelno AS VARCHAR) + ', Type: ' + @Type;

FETCH NEXT FROM cur\_product INTO @Maker, @Modelno, @Type;

END

CLOSE cur\_product;

DEALLOCATE cur\_product;

-- DOCTOR table

CREATE TABLE DOCTOR (

Did INT PRIMARY KEY,

Dname VARCHAR(50) NOT NULL,

Daddress VARCHAR(100) NOT NULL,

qualification VARCHAR(50) NOT NULL

);

-- PATIENTMASTER table

CREATE TABLE PATIENTMASTER (

Pcode INT PRIMARY KEY,

Pname VARCHAR(50) NOT NULL,

Padd VARCHAR(100) NOT NULL,

age INT NOT NULL,

gender CHAR(1) NOT NULL CHECK (gender IN ('M', 'F')),

bloodgroup VARCHAR(2) NOT NULL,

Pid INT NOT NULL,

FOREIGN KEY (Pid) REFERENCES DOCTOR(Did)

);

-- ADMITTEDPATIENT table

CREATE TABLE ADMITTEDPATIENT (

P\_code INT PRIMARY KEY,

Entry\_date DATE NOT NULL,

Discharge\_date DATE NOT NULL,

wardno INT NOT NULL CHECK (wardno < 6),

disease VARCHAR(50) NOT NULL,

FOREIGN KEY (P\_code) REFERENCES PATIENTMASTER(Pcode)

);

-- DOCTOR

INSERT INTO DOCTOR VALUES

(1, 'Dr. Mehta', 'Nashik', 'MBBS'),

(2, 'Dr. Sharma', 'Mumbai', 'MD'),

(3, 'Dr. Gupta', 'Pune', 'MS'),

(4, 'Dr. Roy', 'Delhi', 'MD'),

(5, 'Dr. Joshi', 'Nagpur', 'MBBS'),

(6, 'Dr. Khan', 'Aurangabad', 'MS'),

(7, 'Dr. Patel', 'Surat', 'MBBS'),

(8, 'Dr. Shah', 'Ahmedabad', 'MD'),

(9, 'Dr. Dixit', 'Indore', 'MBBS'),

(10, 'Dr. Rao', 'Hyderabad', 'MS');

-- PATIENTMASTER

INSERT INTO PATIENTMASTER VALUES

(101, 'Anil', 'Nashik', 30, 'M', 'A', 1),

(102, 'Sunita', 'Mumbai', 25, 'F', 'B', 2),

(103, 'Ravi', 'Pune', 40, 'M', 'AB', 3),

(104, 'Meena', 'Delhi', 35, 'F', 'O', 4),

(105, 'Ajay', 'Nagpur', 45, 'M', 'B', 5),

(106, 'Rita', 'Aurangabad', 28, 'F', 'A', 6),

(107, 'Sohan', 'Surat', 22, 'M', 'O', 7),

(108, 'Neha', 'Ahmedabad', 19, 'F', 'AB', 8),

(109, 'Vijay', 'Indore', 50, 'M', 'B', 9),

(110, 'Geeta', 'Hyderabad', 33, 'F', 'A', 10);

-- ADMITTEDPATIENT

INSERT INTO ADMITTEDPATIENT VALUES

(101, '2012-03-01', '2012-03-10', 3, 'Flu'),

(102, '2012-03-05', '2012-03-15', 2, 'Fever'),

(103, '2012-03-10', '2012-03-20', 4, 'Infection'),

(104, '2012-02-25', '2012-03-05', 1, 'Allergy'),

(105, '2012-03-08', '2012-03-12', 3, 'Covid'),

(106, '2012-03-03', '2012-03-09', 2, 'Injury'),

(107, '2012-03-01', '2012-03-06', 1, 'Malaria'),

(108, '2012-03-03', '2012-03-08', 5, 'Fever'),

(109, '2012-03-07', '2012-03-16', 4, 'Cough'),

(110, '2012-03-02', '2012-03-10', 3, 'Typhoid');

**a) Find the details of doctors treating patients in ward no. 3**

SELECT D.\*, P.Pname, A.disease

FROM DOCTOR D

JOIN PATIENTMASTER P ON D.Did = P.Pid

JOIN ADMITTEDPATIENT A ON P.Pcode = A.P\_code

WHERE A.wardno = 3;

**b) Trigger on PATIENTMASTER: blood group must be A, B, AB, or O**

CREATE TRIGGER trg\_check\_bloodgroup

ON PATIENTMASTER

INSTEAD OF INSERT

AS

BEGIN

IF EXISTS (

SELECT \* FROM inserted WHERE bloodgroup NOT IN ('A', 'B', 'AB', 'O')

)

BEGIN

RAISERROR('Invalid blood group. Allowed: A, B, AB, O.', 16, 1);

RETURN;

END

ELSE

BEGIN

INSERT INTO PATIENTMASTER (Pcode, Pname, Padd, age, gender, bloodgroup, Pid)

SELECT Pcode, Pname, Padd, age, gender, bloodgroup, Pid FROM inserted;

END

END;

**c) Find patient details discharged between 03/03/2012 and 25/03/2012**

SELECT P.\*, A.Entry\_date, A.Discharge\_date, A.wardno, A.disease

FROM PATIENTMASTER P

JOIN ADMITTEDPATIENT A ON P.Pcode = A.P\_code

WHERE A.Discharge\_date BETWEEN '2012-03-03' AND '2012-03-25';

**d) Procedure to calculate bill for all discharged patients**

CREATE PROCEDURE CalculatePatientBills

AS

BEGIN

SELECT

P.Pcode,

P.Pname,

A.Entry\_date,

A.Discharge\_date,

A.wardno,

DATEDIFF(DAY, A.Entry\_date, A.Discharge\_date) AS Days\_Admitted,

(DATEDIFF(DAY, A.Entry\_date, A.Discharge\_date) \* A.wardno \* 100) AS BillAmount

FROM PATIENTMASTER P

JOIN ADMITTEDPATIENT A ON P.Pcode = A.P\_code;

END;

-- Execute the procedure

EXEC CalculatePatientBills;

**✅ a) Find the details of the doctors who are treating the patients of ward no 3**

SELECT

D.Did,

D.Dname,

D.Daddress,

D.qualification,

P.Pname AS PatientName,

A.disease

FROM DOCTOR D

JOIN PATIENTMASTER P ON D.Did = P.Pid

JOIN ADMITTEDPATIENT A ON P.Pcode = A.P\_code

WHERE A.wardno = 3;

**✅ b) Find the name of the disease by which maximum patients are suffering**

SELECT TOP 1

disease,

COUNT(\*) AS patient\_count

FROM ADMITTEDPATIENT

GROUP BY disease

ORDER BY patient\_count DESC;

For **MySQL**, replace TOP 1 with LIMIT 1 at the end.

**✅ c) Trigger to ensure wardno is between 1–5**

CREATE TRIGGER trg\_wardno\_check

ON ADMITTEDPATIENT

INSTEAD OF INSERT

AS

BEGIN

IF EXISTS (SELECT \* FROM inserted WHERE wardno < 1 OR wardno > 5)

BEGIN

RAISERROR('Ward number must be between 1 and 5.', 16, 1);

RETURN;

END

ELSE

BEGIN

INSERT INTO ADMITTEDPATIENT (P\_code, Entry\_date, Discharge\_date, wardno, disease)

SELECT P\_code, Entry\_date, Discharge\_date, wardno, disease

FROM inserted;

END

END;

**✅ d) Procedure to list patients admitted for more than 5 days**

CREATE PROCEDURE GetLongStayPatients

AS

BEGIN

SELECT

P.Pcode,

P.Pname,

A.Entry\_date,

A.Discharge\_date,

DATEDIFF(DAY, A.Entry\_date, A.Discharge\_date) AS DaysAdmitted

FROM PATIENTMASTER P

JOIN ADMITTEDPATIENT A ON P.Pcode = A.P\_code

WHERE DATEDIFF(DAY, A.Entry\_date, A.Discharge\_date) > 5;

END;

-- To execute:

EXEC GetLongStayPatients;

**a) Find details of the patients treated by M.B.B.S. doctors**

SELECT

P.\*

FROM PATIENTMASTER P

JOIN DOCTOR D ON P.Pid = D.Did

WHERE D.qualification = 'MBBS';

**✅ b) Find patient details who are suffering from 'blood cancer', age < 50, and blood group is 'A'**

SELECT

P.\*,

A.disease

FROM PATIENTMASTER P

JOIN ADMITTEDPATIENT A ON P.Pcode = A.P\_code

WHERE A.disease = 'blood cancer'

AND P.age < 50

AND P.bloodgroup = 'A';

**✅ c) Procedure to calculate bill of all patients**

**Formula**: Bill = number of days \* 600

CREATE PROCEDURE CalculatePatientBills

AS

BEGIN

SELECT

A.P\_code,

P.Pname,

A.Entry\_date,

A.Discharge\_date,

DATEDIFF(DAY, A.Entry\_date, A.Discharge\_date) AS NoOfDays,

DATEDIFF(DAY, A.Entry\_date, A.Discharge\_date) \* 600 AS BillAmount

FROM ADMITTEDPATIENT A

JOIN PATIENTMASTER P ON A.P\_code = P.Pcode;

END;

-- To execute:

EXEC CalculatePatientBills;

**✅ d) Cursor on PATIENTMASTER to fetch and display the last record**

DECLARE @Pcode INT, @Pname VARCHAR(50), @age INT, @bloodgroup VARCHAR(5);

DECLARE patient\_cursor CURSOR FOR

SELECT Pcode, Pname, age, bloodgroup

FROM PATIENTMASTER

ORDER BY Pcode;

OPEN patient\_cursor;

FETCH LAST FROM patient\_cursor INTO @Pcode, @Pname, @age, @bloodgroup;

PRINT 'Last Patient Record:';

PRINT 'Pcode = ' + CAST(@Pcode AS VARCHAR) +

', Pname = ' + @Pname +

', Age = ' + CAST(@age AS VARCHAR) +

', BloodGroup = ' + @bloodgroup;

CLOSE patient\_cursor;

DEALLOCATE patient\_cursor;

**a) Find details of patients who are treated by M.S. doctors**

SELECT P.\*

FROM PATIENTMASTER P

JOIN DOCTOR D ON P.Pid = D.Did

WHERE D.qualification = 'M.S.';

**🔹 b) Find the name of doctor treating the maximum number of patients**

SELECT TOP 1 D.Dname, COUNT(P.Pcode) AS TotalPatients

FROM DOCTOR D

JOIN PATIENTMASTER P ON D.Did = P.Pid

GROUP BY D.Dname

ORDER BY COUNT(P.Pcode) DESC;

✅ For **MySQL**, use:

sql

CopyEdit

SELECT D.Dname, COUNT(P.Pcode) AS TotalPatients

FROM DOCTOR D

JOIN PATIENTMASTER P ON D.Did = P.Pid

GROUP BY D.Dname

ORDER BY TotalPatients DESC

LIMIT 1;

**🔹 c) Procedure: Show patients admitted for more than 15 days**

CREATE PROCEDURE GetLongStayPatients15Days

AS

BEGIN

SELECT

P.Pcode,

P.Pname,

A.Entry\_date,

A.Discharge\_date,

DATEDIFF(DAY, A.Entry\_date, A.Discharge\_date) AS StayDuration

FROM PATIENTMASTER P

JOIN ADMITTEDPATIENT A ON P.Pcode = A.P\_code

WHERE DATEDIFF(DAY, A.Entry\_date, A.Discharge\_date) > 15;

END;

-- To run the procedure:

EXEC GetLongStayPatients15Days;

**🔹 d) Create a view on DOCTOR and PATIENTMASTER, then update patients treated by 'B.A.-M.S.' doctors to 'M.B.B.S.'**

**1. Create the view**

CREATE VIEW DoctorPatientView AS

SELECT

D.Did,

D.Dname,

D.qualification,

P.Pcode,

P.Pname,

P.Pid

FROM DOCTOR D

JOIN PATIENTMASTER P ON D.Did = P.Pid;

**2. Update DOCTOR's qualification from 'B.A.-M.S.' to 'M.B.B.S.'**

UPDATE DOCTOR

SET qualification = 'M.B.B.S'

WHERE qualification = 'B.A.-M.S.';

-- ACCOUNT table

CREATE TABLE ACCOUNT (

accno INT PRIMARY KEY CHECK (accno < 1000),

open\_date DATE NOT NULL,

acctype CHAR(1) NOT NULL CHECK (acctype IN ('P', 'J')),

balance DECIMAL(12, 2) NOT NULL

);

-- CUSTOMER table

CREATE TABLE CUSTOMER (

cust\_id INT PRIMARY KEY,

name VARCHAR(50) NOT NULL,

address VARCHAR(100) NOT NULL,

accno INT NOT NULL,

FOREIGN KEY (accno) REFERENCES ACCOUNT(accno)

);

-- TRANSACTION table

CREATE TABLE TRANSACTION (

trans\_id INT PRIMARY KEY,

trans\_date DATE NOT NULL,

accno INT NOT NULL,

trans\_type CHAR(1) NOT NULL CHECK (trans\_type IN ('C', 'D')),

amount DECIMAL(12, 2) NOT NULL,

FOREIGN KEY (accno) REFERENCES ACCOUNT(accno)

);

-- ACCOUNT table

INSERT INTO ACCOUNT VALUES

(101, '2012-01-01', 'P', 150000),

(102, '2012-01-15', 'J', 90000),

(103, '2012-02-01', 'P', 50000),

(104, '2012-02-20', 'J', 120000),

(105, '2012-03-01', 'P', 70000),

(106, '2012-03-10', 'P', 100000),

(107, '2012-03-15', 'J', 40000),

(108, '2012-03-25', 'P', 300000),

(109, '2012-03-26', 'J', 85000),

(110, '2012-03-28', 'P', 110000);

-- CUSTOMER table

INSERT INTO CUSTOMER VALUES

(1, 'Anil', 'Nashik', 101),

(2, 'Sunita', 'Pune', 102),

(3, 'Ravi', 'Mumbai', 103),

(4, 'Kiran', 'Delhi', 104),

(5, 'Ramesh', 'Nagpur', 105),

(6, 'Sita', 'Aurangabad', 106),

(7, 'Neha', 'Ahmedabad', 107),

(8, 'Arjun', 'Nanded', 108),

(9, 'Meena', 'Surat', 109),

(10, 'Deepak', 'Indore', 110);

-- TRANSACTION table

INSERT INTO TRANSACTION VALUES

(1, '2012-03-25', 101, 'C', 5000),

(2, '2012-03-26', 102, 'C', 10000),

(3, '2012-03-27', 108, 'C', 20000),

(4, '2012-03-28', 109, 'C', 8000),

(5, '2012-03-25', 110, 'D', 1000),

(6, '2012-03-20', 101, 'D', 5000),

(7, '2012-03-15', 105, 'C', 7000),

(8, '2012-03-10', 106, 'D', 6000),

(9, '2012-03-12', 104, 'C', 3000),

(10, '2012-03-05', 103, 'D', 2000);

**a) Customers with minimum balance 1 lakh**

SELECT C.\*

FROM CUSTOMER C

JOIN ACCOUNT A ON C.accno = A.accno

WHERE A.balance >= 100000;

**b) Amount credited between 25-03-2012 to 28-03-2012**

SELECT \*

FROM TRANSACTION

WHERE trans\_type = 'C'

AND trans\_date BETWEEN '2012-03-25' AND '2012-03-28';

**c) Trigger to update account balance after a transaction**

CREATE TRIGGER trg\_update\_balance

ON TRANSACTION

AFTER INSERT

AS

BEGIN

UPDATE ACCOUNT

SET balance = balance + I.amount

FROM ACCOUNT A

JOIN inserted I ON A.accno = I.accno

WHERE I.trans\_type = 'C';

UPDATE ACCOUNT

SET balance = balance - I.amount

FROM ACCOUNT A

JOIN inserted I ON A.accno = I.accno

WHERE I.trans\_type = 'D';

END;

**d) Cursor on ACCOUNT to show loan eligibility**

DECLARE @accno INT, @balance DECIMAL(12,2);

DECLARE acc\_cursor CURSOR FOR

SELECT accno, balance FROM ACCOUNT;

OPEN acc\_cursor;

FETCH NEXT FROM acc\_cursor INTO @accno, @balance;

WHILE @@FETCH\_STATUS = 0

BEGIN

IF @balance < 10000

PRINT 'Account ' + CAST(@accno AS VARCHAR) + ': Loan is not provided';

ELSE

PRINT 'Account ' + CAST(@accno AS VARCHAR) + ': Loan is provided';

FETCH NEXT FROM acc\_cursor INTO @accno, @balance;

END;

CLOSE acc\_cursor;

DEALLOCATE acc\_cursor;

**🟢 a) Details of customers with personal accounts & balance < 2 lakhs**

SELECT C.\*

FROM CUSTOMER C

JOIN ACCOUNT A ON C.accno = A.accno

WHERE A.acctype = 'P' AND A.balance < 200000;

**🟢 b) Details of customers with joint accounts**

SELECT C.\*

FROM CUSTOMER C

JOIN ACCOUNT A ON C.accno = A.accno

WHERE A.acctype = 'J';

**🟢 c) Trigger: prevent withdrawal if balance < 300**

This trigger checks for withdrawal attempts (trans\_type = 'D') and cancels them if the balance would go below ₹300:

CREATE TRIGGER trg\_prevent\_low\_balance

ON TRANSACTION

INSTEAD OF INSERT

AS

BEGIN

IF EXISTS (

SELECT 1

FROM inserted I

JOIN ACCOUNT A ON A.accno = I.accno

WHERE I.trans\_type = 'D'

AND A.balance - I.amount < 300

)

BEGIN

RAISERROR('Withdrawal denied: balance cannot go below ₹300.', 16, 1);

RETURN;

END

-- Proceed if allowed

INSERT INTO TRANSACTION (trans\_id, trans\_date, accno, trans\_type, amount)

SELECT trans\_id, trans\_date, accno, trans\_type, amount FROM inserted;

-- Update account balance

UPDATE ACCOUNT

SET balance = balance + I.amount

FROM ACCOUNT A

JOIN inserted I ON A.accno = I.accno

WHERE I.trans\_type = 'C';

UPDATE ACCOUNT

SET balance = balance - I.amount

FROM ACCOUNT A

JOIN inserted I ON A.accno = I.accno

WHERE I.trans\_type = 'D';

END;

**🟢 d) Procedure: add new transaction & update balance**

CREATE PROCEDURE AddTransaction

@trans\_id INT,

@trans\_date DATE,

@accno INT,

@trans\_type CHAR(1),

@amount DECIMAL(12, 2)

AS

BEGIN

-- Insert transaction

INSERT INTO TRANSACTION (trans\_id, trans\_date, accno, trans\_type, amount)

VALUES (@trans\_id, @trans\_date, @accno, @trans\_type, @amount);

-- Update balance

IF @trans\_type = 'C'

BEGIN

UPDATE ACCOUNT

SET balance = balance + @amount

WHERE accno = @accno;

END

ELSE IF @trans\_type = 'D'

BEGIN

-- Optional safeguard

DECLARE @current\_balance DECIMAL(12, 2);

SELECT @current\_balance = balance FROM ACCOUNT WHERE accno = @accno;

IF @current\_balance - @amount >= 300

BEGIN

UPDATE ACCOUNT

SET balance = balance - @amount

WHERE accno = @accno;

END

ELSE

BEGIN

RAISERROR('Transaction denied: balance would drop below ₹300.', 16, 1);

END

END

END;

-- Example usage

-- EXEC AddTransaction 201, '2024-06-11', 101, 'D', 500;

**a) Details of all transactions for accno = 101 + customer names**

sql

CopyEdit

SELECT

T.\*,

C.name AS CustomerName

FROM TRANSACTION T

JOIN CUSTOMER C ON T.accno = C.accno

WHERE T.accno = 101;

**🔹 b) Details of amount credited between 15-03-2012 and 18-03-2012**

sql

CopyEdit

SELECT \*

FROM TRANSACTION

WHERE trans\_type = 'C'

AND trans\_date BETWEEN '2012-03-15' AND '2012-03-18';

**🔹 c) Trigger on ACCOUNT to block debit if balance ≤ 500**

This trigger works during a debit transaction (trans\_type = 'D') and stops it if balance is too low:

sql

CopyEdit

CREATE TRIGGER trg\_prevent\_debit\_low\_balance

ON TRANSACTION

INSTEAD OF INSERT

AS

BEGIN

IF EXISTS (

SELECT 1

FROM inserted I

JOIN ACCOUNT A ON I.accno = A.accno

WHERE I.trans\_type = 'D' AND A.balance <= 500

)

BEGIN

RAISERROR('Debit not allowed: balance is less than or equal to ₹500.', 16, 1);

RETURN;

END

-- Insert valid transaction

INSERT INTO TRANSACTION (trans\_id, trans\_date, accno, trans\_type, amount)

SELECT trans\_id, trans\_date, accno, trans\_type, amount FROM inserted;

-- Update balance

UPDATE ACCOUNT

SET balance = balance + I.amount

FROM ACCOUNT A

JOIN inserted I ON A.accno = I.accno

WHERE I.trans\_type = 'C';

UPDATE ACCOUNT

SET balance = balance - I.amount

FROM ACCOUNT A

JOIN inserted I ON A.accno = I.accno

WHERE I.trans\_type = 'D';

END;

**🔹 d) Procedure to calculate interest from open\_date to today**

sql

CopyEdit

CREATE PROCEDURE CalculateInterest

@accno INT,

@interest\_rate FLOAT -- e.g. pass 5 for 5%

AS

BEGIN

DECLARE @open\_date DATE, @balance DECIMAL(12,2), @days INT, @interest DECIMAL(12,2);

SELECT @open\_date = open\_date, @balance = balance

FROM ACCOUNT

WHERE accno = @accno;

SET @days = DATEDIFF(DAY, @open\_date, GETDATE());

SET @interest = (@balance \* @interest\_rate \* @days) / (100 \* 365.0); -- Simple interest

PRINT 'Account Number: ' + CAST(@accno AS VARCHAR);

PRINT 'Balance: ₹' + CAST(@balance AS VARCHAR);

PRINT 'Interest for ' + CAST(@days AS VARCHAR) + ' days at ' + CAST(@interest\_rate AS VARCHAR) + '% = ₹' + CAST(@interest AS VARCHAR);

END;

-- Example:

-- EXEC CalculateInterest @accno = 101, @interest\_rate = 5;

**a) Find customers who opened accounts between 25-03-2012 and 28-03-2012**

sql

CopyEdit

SELECT C.\*

FROM CUSTOMER C

JOIN ACCOUNT A ON C.accno = A.accno

WHERE A.open\_date BETWEEN '2012-03-25' AND '2012-03-28';

**🔹 b) Find customers with joint accounts and balance less than ₹2,00,000**

sql

CopyEdit

SELECT C.\*

FROM CUSTOMER C

JOIN ACCOUNT A ON C.accno = A.accno

WHERE A.acctype = 'J' AND A.balance < 200000;

**🔹 c) Trigger on TRANSACTION table to update account balance**

sql

CopyEdit

CREATE TRIGGER trg\_update\_balance\_after\_transaction

ON TRANSACTION

AFTER INSERT

AS

BEGIN

-- Credit case

UPDATE ACCOUNT

SET balance = balance + I.amount

FROM ACCOUNT A

JOIN inserted I ON A.accno = I.accno

WHERE I.trans\_type = 'C';

-- Debit case

UPDATE ACCOUNT

SET balance = balance - I.amount

FROM ACCOUNT A

JOIN inserted I ON A.accno = I.accno

WHERE I.trans\_type = 'D';

END;

**🔹 d) Cursor on CUSTOMER table to fetch the last row**

DECLARE @cust\_id INT, @name VARCHAR(50), @address VARCHAR(100), @accno INT;

DECLARE customer\_cursor CURSOR FOR

SELECT cust\_id, name, address, accno

FROM CUSTOMER

ORDER BY cust\_id;

OPEN customer\_cursor;

FETCH LAST FROM customer\_cursor INTO @cust\_id, @name, @address, @accno;

PRINT 'Last Customer Record:';

PRINT 'ID: ' + CAST(@cust\_id AS VARCHAR) + ', Name: ' + @name +

', Address: ' + @address + ', Account No: ' + CAST(@accno AS VARCHAR);

CLOSE customer\_cursor;

DEALLOCATE customer\_cursor;

✅ If your SQL engine **does not support FETCH LAST**, use this instead:

sql

CopyEdit

SELECT TOP 1 \*

FROM CUSTOMER

ORDER BY cust\_id DESC;

-- DEPT Table

CREATE TABLE DEPT (

dname VARCHAR(50) NOT NULL,

dnum INT PRIMARY KEY CHECK (dnum < 10000),

mgrssn CHAR(9) NOT NULL,

dlocation VARCHAR(50) NOT NULL

);

-- EMPLOYEE Table

CREATE TABLE EMPLOYEE (

fname VARCHAR(50) NOT NULL,

mname VARCHAR(50) NOT NULL,

ssn CHAR(9) PRIMARY KEY,

sex CHAR(1) NOT NULL,

salary DECIMAL(10, 2) NOT NULL,

joindate DATE NOT NULL,

superssn CHAR(9),

dno INT NOT NULL,

FOREIGN KEY (dno) REFERENCES DEPT(dnum),

FOREIGN KEY (superssn) REFERENCES EMPLOYEE(ssn)

);

-- PROJECT Table

CREATE TABLE PROJECT (

pname VARCHAR(50) NOT NULL,

pno INT PRIMARY KEY,

plocation VARCHAR(50) NOT NULL,

dnumber INT NOT NULL,

FOREIGN KEY (dnumber) REFERENCES DEPT(dnum)

);

-- WORKS\_ON Table

CREATE TABLE WORKS\_ON (

ssn CHAR(9) NOT NULL,

pno INT NOT NULL,

hours DECIMAL(5, 2) NOT NULL,

FOREIGN KEY (ssn) REFERENCES EMPLOYEE(ssn),

FOREIGN KEY (pno) REFERENCES PROJECT(pno)

);

SELECT

P.pno,

P.dnumber,

E.lname AS ManagerLastName

FROM PROJECT P

JOIN DEPT D ON P.dnumber = D.dnum

JOIN EMPLOYEE E ON D.mgrssn = E.ssn

WHERE P.plocation = 'Jalgaon';

**🔹 a) Projects located in 'Jalgaon': pno, dept number, and manager’s last name**

SELECT

P.pno,

P.dnumber,

E.lname AS ManagerLastName

FROM PROJECT P

JOIN DEPT D ON P.dnumber = D.dnum

JOIN EMPLOYEE E ON D.mgrssn = E.ssn

WHERE P.plocation = 'Jalgaon';

If lname isn’t available, you can use fname or add lname to the schema.

**🔹 b) Projects with more than 2 employees: pno, pname, count**

SELECT

P.pno,

P.pname,

COUNT(W.ssn) AS emp\_count

FROM PROJECT P

JOIN WORKS\_ON W ON P.pno = W.pno

GROUP BY P.pno, P.pname

HAVING COUNT(W.ssn) > 2;

**🔹 c) Create a view with deptname, manager name, and salary**

CREATE VIEW DeptManagerView AS

SELECT

D.dname,

CONCAT(E.fname, ' ', E.mname) AS ManagerName,

E.salary AS ManagerSalary

FROM DEPT D

JOIN EMPLOYEE E ON D.mgrssn = E.ssn;

**🔹 d) SQL Assertion: salary of employee ≤ salary of manager**

CREATE ASSERTION salary\_check

CHECK (

NOT EXISTS (

SELECT \*

FROM EMPLOYEE E

JOIN EMPLOYEE M ON E.dno = M.dno AND M.ssn = (

SELECT mgrssn FROM DEPT WHERE dnum = E.dno

)

WHERE E.salary > M.salary

)

);

**🔹 a) For each employee, show their name & their immediate supervisor's name**

sql

CopyEdit

SELECT

E.fname AS Emp\_FirstName,

E.mname AS Emp\_LastName,

S.fname AS Sup\_FirstName,

S.mname AS Sup\_LastName

FROM EMPLOYEE E

LEFT JOIN EMPLOYEE S ON E.superssn = S.ssn;

**🔹 b) For each department: deptno, number of employees & average salary**

sql

CopyEdit

SELECT

dno AS DeptNo,

COUNT(\*) AS NumEmployees,

AVG(salary) AS AvgSalary

FROM EMPLOYEE

GROUP BY dno;

**🔹 c) Create a view: project name, dept name, no. of employees, total hours per week – for projects with > 1 employee**

sql

CopyEdit

CREATE VIEW ProjectSummaryView AS

SELECT

P.pname,

D.dname AS DeptName,

COUNT(W.ssn) AS NumEmployees,

SUM(W.hours) AS TotalHoursPerWeek

FROM PROJECT P

JOIN DEPT D ON P.dnumber = D.dnum

JOIN WORKS\_ON W ON P.pno = W.pno

GROUP BY P.pname, D.dname

HAVING COUNT(W.ssn) > 1;

**🔹 d) Procedure: Find employees working on 5+ projects (eligible for promotion)**

CREATE PROCEDURE GetPromotableEmployees

AS

BEGIN

SELECT

E.fname,

E.mname,

E.ssn,

COUNT(DISTINCT W.pno) AS ProjectsWorkedOn

FROM EMPLOYEE E

JOIN WORKS\_ON W ON E.ssn = W.ssn

GROUP BY E.fname, E.mname, E.ssn

HAVING COUNT(DISTINCT W.pno) >= 5;

END;

-- Usage:

-- EXEC GetPromotableEmployees;

**a) Find SSNs of all employees who work on project no. 101, 102, or 103**

sql

CopyEdit

SELECT DISTINCT ssn

FROM WORKS\_ON

WHERE pno IN (101, 102, 103);

**✅ b) List all project numbers where an employee with last name 'sonar' is either:**

* Working **on** the project
* OR Managing the **department** that controls the project

Assuming lname is stored in the mname column (since schema has no lname, but you might be using mname for that):

sql

CopyEdit

-- Step 1: Employees named 'Sonar' working on projects

SELECT DISTINCT W.pno

FROM WORKS\_ON W

JOIN EMPLOYEE E ON W.ssn = E.ssn

WHERE E.mname = 'Sonar'

UNION

-- Step 2: Projects managed by someone named 'Sonar'

SELECT DISTINCT P.pno

FROM PROJECT P

JOIN DEPT D ON P.dnumber = D.dnum

JOIN EMPLOYEE M ON D.mgrssn = M.ssn

WHERE M.mname = 'Sonar';

**✅ c) Trigger: On insert into WORKS\_ON, deduct salary if total hours < 20**

This assumes we are checking the **total working hours** of an employee **after** the new insert.

sql

CopyEdit

CREATE TRIGGER trg\_deduct\_salary\_if\_less\_than\_20hrs

ON WORKS\_ON

AFTER INSERT

AS

BEGIN

DECLARE @ssn CHAR(9), @total\_hours DECIMAL(5, 2);

SELECT @ssn = ssn FROM inserted;

SELECT @total\_hours = SUM(hours)

FROM WORKS\_ON

WHERE ssn = @ssn;

IF @total\_hours < 20

BEGIN

-- Deduct 10% of salary for example

UPDATE EMPLOYEE

SET salary = salary \* 0.9

WHERE ssn = @ssn;

END

END;

🔎 You can modify the percentage deduction logic as per policy.

**✅ d) Cursor on PROJECT to fetch the first row & total number of rows**

sql

CopyEdit

DECLARE @pname VARCHAR(50), @pno INT, @count INT;

-- Count total rows

SELECT @count = COUNT(\*) FROM PROJECT;

-- Declare cursor

DECLARE project\_cursor CURSOR FOR

SELECT pname, pno FROM PROJECT ORDER BY pno;

OPEN project\_cursor;

-- Fetch first row

FETCH NEXT FROM project\_cursor INTO @pname, @pno;

-- Display

PRINT 'First Project: ' + @pname + ' (Project No: ' + CAST(@pno AS VARCHAR) + ')';

PRINT 'Total Projects: ' + CAST(@count AS VARCHAR);

-- Close and deallocate

CLOSE project\_cursor;

DEALLOCATE project\_cursor;

CREATE TABLE BOOKMASTER (

bid INT PRIMARY KEY,

title VARCHAR(100) NOT NULL,

author VARCHAR(100) NOT NULL,

price DECIMAL(6,2) NOT NULL

);

CREATE TABLE STUDENTMASTER (

enrollno INT PRIMARY KEY,

sname VARCHAR(100) NOT NULL,

class VARCHAR(50) NOT NULL,

dept VARCHAR(50) NOT NULL

);

CREATE TABLE ACCESSIONTABLE (

accession\_no INT PRIMARY KEY,

bid INT NOT NULL,

avail CHAR(1) CHECK (avail IN ('T', 'F')),

FOREIGN KEY (bid) REFERENCES BOOKMASTER(bid)

);

CREATE TABLE ISSUETABLE (

issueid INT PRIMARY KEY,

accession\_no INT NOT NULL,

enrollno INT NOT NULL,

issuedate DATE NOT NULL,

duedate DATE NOT NULL,

ret\_date DATE,

bid INT NOT NULL,

FOREIGN KEY (accession\_no) REFERENCES ACCESSIONTABLE(accession\_no),

FOREIGN KEY (enrollno) REFERENCES STUDENTMASTER(enrollno),

FOREIGN KEY (bid) REFERENCES BOOKMASTER(bid)

);

-- BOOKMASTER

INSERT INTO BOOKMASTER VALUES

(1, 'DBMS', 'Korth', 500),

(2, 'OS', 'Galvin', 600),

(3, 'CN', 'Tanenbaum', 700),

(4, 'Java Basics', 'Herbert', 450),

(5, 'AI', 'Russell', 750),

(6, 'Python', 'Mark Lutz', 550),

(7, 'DSA', 'Sahni', 670),

(8, 'ML Basics', 'Goodfellow', 720),

(9, 'Computer Graphics', 'Hearn', 580),

(10, 'Cryptography', 'Stallings', 640);

-- STUDENTMASTER

INSERT INTO STUDENTMASTER VALUES

(101, 'Ravi', 'TYBCA', 'Computer'),

(102, 'Priya', 'SYBCA', 'Computer'),

(103, 'Anil', 'FYBCA', 'Computer'),

(104, 'Neha', 'TYBBA', 'Management'),

(105, 'Amit', 'SYBBA', 'Management'),

(106, 'Kiran', 'FYBCA', 'Computer'),

(107, 'Tina', 'SYBCA', 'Computer'),

(108, 'Raj', 'TYBBA', 'Management'),

(109, 'Sneha', 'FYBCA', 'Computer'),

(110, 'Deepak', 'TYBCA', 'Computer');

-- ACCESSIONTABLE

INSERT INTO ACCESSIONTABLE VALUES

(1001, 1, 'T'), (1002, 2, 'F'), (1003, 3, 'T'),

(1004, 4, 'F'), (1005, 5, 'T'), (1006, 6, 'T'),

(1007, 7, 'F'), (1008, 8, 'T'), (1009, 9, 'F'), (1010, 10, 'T');

-- ISSUETABLE

INSERT INTO ISSUETABLE VALUES

(1, 1002, 101, '2025-06-01', '2025-06-08', '2025-06-09', 2),

(2, 1004, 102, '2025-06-02', '2025-06-09', '2025-06-11', 4),

(3, 1007, 103, '2025-06-03', '2025-06-10', '2025-06-10', 7),

(4, 1009, 106, '2025-06-01', '2025-06-08', '2025-06-12', 9);

**a) Find the name of books which is issued maximum times**

SELECT b.title, COUNT(\*) AS times\_issued

FROM ISSUETABLE i

JOIN BOOKMASTER b ON i.bid = b.bid

GROUP BY i.bid

ORDER BY times\_issued DESC

LIMIT 1;

**b) Details of books issued by Computer department students**

sql

CopyEdit

SELECT b.\*

FROM ISSUETABLE i

JOIN STUDENTMASTER s ON i.enrollno = s.enrollno

JOIN BOOKMASTER b ON i.bid = b.bid

WHERE s.dept = 'Computer';

**4. Procedure to Calculate Fines**

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE CalculateFines()

BEGIN

SELECT

i.issueid,

i.enrollno,

DATEDIFF(i.ret\_date, i.duedate) AS days\_late,

CASE

WHEN i.ret\_date > i.duedate THEN DATEDIFF(i.ret\_date, i.duedate) \* 10

ELSE 0

END AS fine

FROM ISSUETABLE i

WHERE i.ret\_date IS NOT NULL;

END //

DELIMITER ;

Run it using:

sql

CopyEdit

CALL CalculateFines();

**5. Trigger to Auto-Set duedate = issuedate + 7**

sql

CopyEdit

DELIMITER //

CREATE TRIGGER set\_duedate\_before\_insert

BEFORE INSERT ON ISSUETABLE

FOR EACH ROW

BEGIN

SET NEW.duedate = DATE\_ADD(NEW.issuedate, INTERVAL 7 DAY);

END //

DELIMITER ;

**a) Find the detail information of the students who have issued books between two given dates**

sql

CopyEdit

-- Replace '2025-06-01' and '2025-06-10' with your desired date range

SELECT s.\*

FROM ISSUETABLE i

JOIN STUDENTMASTER s ON i.enrollno = s.enrollno

WHERE i.issuedate BETWEEN '2025-06-01' AND '2025-06-10';

**b) Create a view that displays all the accession information for a book having bid = 100**

sql

CopyEdit

CREATE VIEW View\_Accession\_Info AS

SELECT \*

FROM ACCESSIONTABLE

WHERE bid = 100;

To view the result:

sql

CopyEdit

SELECT \* FROM View\_Accession\_Info;

**c) Write a cursor to fetch the last record from the view in (b)**

Note: MySQL cursors do not directly support reverse traversal. However, you can simulate fetching the "last" record by ordering and limiting.

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE FetchLastAccessionRecord()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE acc\_no INT;

DECLARE book\_id INT;

DECLARE available CHAR(1);

DECLARE cur CURSOR FOR

SELECT accession\_no, bid, avail

FROM View\_Accession\_Info

ORDER BY accession\_no DESC

LIMIT 1;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO acc\_no, book\_id, available;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT acc\_no AS accession\_no, book\_id AS bid, available AS avail;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

Call the procedure:

sql

CopyEdit

CALL FetchLastAccessionRecord();

**d) Find the information of books issued by MCA students**

sql

CopyEdit

SELECT DISTINCT b.\*

FROM ISSUETABLE i

JOIN STUDENTMASTER s ON i.enrollno = s.enrollno

JOIN BOOKMASTER b ON i.bid = b.bid

WHERE s.dept = 'MCA';

**a) Procedure to show available books in the library**

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE ShowAvailableBooks()

BEGIN

SELECT b.\*

FROM ACCESSIONTABLE a

JOIN BOOKMASTER b ON a.bid = b.bid

WHERE a.avail = 'T';

END //

DELIMITER ;

-- To run:

CALL ShowAvailableBooks();

**✅ b) Find number of books issued by each student**

sql

CopyEdit

SELECT s.enrollno, s.sname, COUNT(i.issueid) AS books\_issued

FROM STUDENTMASTER s

LEFT JOIN ISSUETABLE i ON s.enrollno = i.enrollno

GROUP BY s.enrollno, s.sname;

**✅ c) Trigger: Return date must not exceed today’s date**

sql

CopyEdit

DELIMITER //

CREATE TRIGGER check\_return\_date

BEFORE INSERT ON ISSUETABLE

FOR EACH ROW

BEGIN

IF NEW.ret\_date > CURDATE() THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Return date cannot be in the future.';

END IF;

END //

DELIMITER ;

**✅ d) Find number of available books by “Henry Korth”**

sql

CopyEdit

SELECT COUNT(\*) AS available\_books

FROM ACCESSIONTABLE a

JOIN BOOKMASTER b ON a.bid = b.bid

WHERE a.avail = 'T' AND b.author = 'Henry Korth';

**✅ e) Class-wise issue report**

sql

CopyEdit

SELECT s.class, COUNT(i.issueid) AS total\_books\_issued

FROM STUDENTMASTER s

JOIN ISSUETABLE i ON s.enrollno = i.enrollno

GROUP BY s.class;

CREATE TABLE ACCOUNT (

accno INT CHECK (accno < 1000) PRIMARY KEY,

open\_date DATE NOT NULL,

acctype CHAR(1) CHECK (acctype IN ('P', 'J')) NOT NULL,

balance DECIMAL(10, 2) NOT NULL

);

CREATE TABLE CUSTOMER (

cust\_id INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

address VARCHAR(255) NOT NULL,

accno INT NOT NULL,

FOREIGN KEY (accno) REFERENCES ACCOUNT(accno)

);

CREATE TABLE TRANSACTION (

trans\_id INT PRIMARY KEY,

trans\_date DATE NOT NULL,

accno INT NOT NULL,

trans\_type CHAR(1) CHECK (trans\_type IN ('C', 'D')) NOT NULL,

amount DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (accno) REFERENCES ACCOUNT(accno)

);

INSERT INTO ACCOUNT VALUES

(101, '2006-03-25', 'P', 300000),

(102, '2006-03-26', 'J', 150000),

(103, '2006-03-27', 'P', 250000),

(104, '2006-03-28', 'J', 100000),

(105, '2006-03-24', 'P', 400000),

(106, '2006-03-29', 'J', 90000),

(107, '2006-03-25', 'P', 700000),

(108, '2006-03-26', 'J', 180000),

(109, '2006-03-27', 'P', 160000),

(110, '2006-03-30', 'J', 210000);

INSERT INTO CUSTOMER VALUES

(1, 'Ravi', 'Mumbai', 101),

(2, 'Neha', 'Pune', 102),

(3, 'Anil', 'Delhi', 103),

(4, 'Pooja', 'Nashik', 104),

(5, 'Kiran', 'Nagpur', 105),

(6, 'Amit', 'Solapur', 106),

(7, 'Sneha', 'Pune', 107),

(8, 'Raj', 'Aurangabad', 108),

(9, 'Tina', 'Mumbai', 109),

(10, 'Deepak', 'Pune', 110);

INSERT INTO TRANSACTION VALUES

(1, '2024-06-01', 101, 'C', 10000),

(2, '2024-06-02', 102, 'D', 5000),

(3, '2024-06-03', 103, 'C', 20000),

(4, '2024-06-04', 104, 'D', 15000),

(5, '2024-06-05', 105, 'C', 30000),

(6, '2024-06-06', 106, 'D', 4000),

(7, '2024-06-07', 107, 'C', 50000),

(8, '2024-06-08', 108, 'D', 10000),

(9, '2024-06-09', 109, 'C', 20000),

(10, '2024-06-10', 110, 'D', 5000);

**✅ a) Customers who opened accounts between 25-03-2006 and 28-03-2006**

sql

CopyEdit

SELECT c.\*

FROM CUSTOMER c

JOIN ACCOUNT a ON c.accno = a.accno

WHERE a.open\_date BETWEEN '2006-03-25' AND '2006-03-28';

**✅ b) Customers with Joint accounts and balance < 2 lakhs**

sql

CopyEdit

SELECT c.\*

FROM CUSTOMER c

JOIN ACCOUNT a ON c.accno = a.accno

WHERE a.acctype = 'J' AND a.balance < 200000;

**⚙️ Triggers & Cursors**

**✅ c) Trigger: Update ACCOUNT balance on TRANSACTION insert**

sql

CopyEdit

DELIMITER //

CREATE TRIGGER update\_balance\_on\_transaction

AFTER INSERT ON TRANSACTION

FOR EACH ROW

BEGIN

IF NEW.trans\_type = 'C' THEN

UPDATE ACCOUNT SET balance = balance + NEW.amount WHERE accno = NEW.accno;

ELSEIF NEW.trans\_type = 'D' THEN

UPDATE ACCOUNT SET balance = balance - NEW.amount WHERE accno = NEW.accno;

END IF;

END //

DELIMITER ;

**✅ d) Cursor to fetch the last record from CUSTOMER table**

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE GetLastCustomer()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE cid INT;

DECLARE cname VARCHAR(100);

DECLARE caddr VARCHAR(255);

DECLARE cacc INT;

DECLARE cur CURSOR FOR

SELECT cust\_id, name, address, accno FROM CUSTOMER ORDER BY cust\_id DESC LIMIT 1;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO cid, cname, caddr, cacc;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT cid AS cust\_id, cname AS name, caddr AS address, cacc AS accno;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

-- Call procedure

CALL GetLastCustomer();

CREATE TABLE PRODUCT (

maker VARCHAR(50) NOT NULL,

modelno INT PRIMARY KEY,

type VARCHAR(10) CHECK (type IN ('PC', 'Laptop', 'Printer')) NOT NULL

);

CREATE TABLE PC (

modelno INT PRIMARY KEY,

speed DECIMAL(4,1) NOT NULL, -- MHz

ram INT NOT NULL, -- MB

hd INT CHECK (hd > 20) NOT NULL, -- GB

cd VARCHAR(10) NOT NULL,

price DECIMAL(10,2) NOT NULL,

FOREIGN KEY (modelno) REFERENCES PRODUCT(modelno)

);

CREATE TABLE LAPTOP (

modelno INT PRIMARY KEY,

speed DECIMAL(4,1) NOT NULL,

ram INT NOT NULL,

hd INT CHECK (hd > 20) NOT NULL,

price DECIMAL(10,2) NOT NULL,

FOREIGN KEY (modelno) REFERENCES PRODUCT(modelno)

);

CREATE TABLE PRINTER (

modelno INT PRIMARY KEY,

color CHAR(1) CHECK (color IN ('T', 'F')) NOT NULL,

type VARCHAR(15) CHECK (type IN ('laser', 'ink-jet', 'dot-matrix', 'dry')) NOT NULL,

price DECIMAL(10,2) NOT NULL,

FOREIGN KEY (modelno) REFERENCES PRODUCT(modelno)

);

**a) Find the manufacturers of color printers**

sql

CopyEdit

SELECT DISTINCT p.maker

FROM PRODUCT p

JOIN PRINTER pr ON p.modelno = pr.modelno

WHERE pr.color = 'T';

**b) Find laptops whose speed is slower than any PC**

sql

CopyEdit

SELECT l.\*

FROM LAPTOP l

WHERE l.speed < ALL (SELECT speed FROM PC);

**c) SQL Assertion: No black & white printer should cost more than any color printer**

sql

CopyEdit

-- Note: Many DBMS (like MySQL) do not support ASSERTION. Below is ANSI SQL style.

CREATE ASSERTION no\_bw\_costlier\_than\_color

CHECK (

NOT EXISTS (

SELECT \*

FROM PRINTER bw, PRINTER color

WHERE bw.color = 'F' AND color.color = 'T' AND bw.price > color.price

)

);

**d) Trigger: Hard disk size must be greater than 20 GB for PC and LAPTOP**

sql

CopyEdit

DELIMITER //

CREATE TRIGGER check\_pc\_hd

BEFORE INSERT ON PC

FOR EACH ROW

BEGIN

IF NEW.hd <= 20 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Hard disk size for PC must be greater than 20 GB';

END IF;

END //

CREATE TRIGGER check\_laptop\_hd

BEFORE INSERT ON LAPTOP

FOR EACH ROW

BEGIN

IF NEW.hd <= 20 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Hard disk size for Laptop must be greater than 20 GB';

END IF;

END //

DELIMITER ;

**✅ a) Find the manufacturers of color printers**

sql

CopyEdit

SELECT DISTINCT p.maker

FROM PRODUCT p

JOIN PRINTER pr ON p.modelno = pr.modelno

WHERE pr.color = 'T';

* This retrieves all **distinct makers** from the PRODUCT table who manufacture **color printers** (i.e., color = 'T').

**✅ b) Find the laptops whose speed is slower than that of any PC**

sql

CopyEdit

SELECT \*

FROM LAPTOP

WHERE speed < ALL (SELECT speed FROM PC);

* This query returns laptops whose **speed is less than the speed of every PC** listed in the PC table.

**✅ c) SQL Assertion: No black & white printer should be more expensive than any color printer**

Note: **MySQL** and many DBMS **do not support CREATE ASSERTION**, but here's the ANSI SQL version:

sql

CopyEdit

CREATE ASSERTION no\_bw\_costlier\_than\_color

CHECK (

NOT EXISTS (

SELECT \*

FROM PRINTER bw, PRINTER color

WHERE bw.color = 'F' AND color.color = 'T' AND bw.price > color.price

)

);

**💡 Workaround for MySQL: Use a BEFORE INSERT/UPDATE Trigger on PRINTER:**

sql

CopyEdit

DELIMITER //

CREATE TRIGGER prevent\_expensive\_bw\_printer

BEFORE INSERT ON PRINTER

FOR EACH ROW

BEGIN

IF NEW.color = 'F' AND

EXISTS (

SELECT 1 FROM PRINTER WHERE color = 'T' AND NEW.price > price

) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Black & white printer cannot be more expensive than color printer';

END IF;

END //

DELIMITER ;

You can also replicate this for BEFORE UPDATE if needed.

**✅ d) Triggers on PC & LAPTOP to enforce HD > 20 GB**

**Trigger for PC:**

sql

CopyEdit

DELIMITER //

CREATE TRIGGER check\_pc\_hd

BEFORE INSERT ON PC

FOR EACH ROW

BEGIN

IF NEW.hd <= 20 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'PC hard disk size must be greater than 20 GB';

END IF;

END //

DELIMITER ;

**a) Find the different types of printers produced by Epson**

sql

CopyEdit

SELECT DISTINCT pr.type

FROM PRODUCT p

JOIN PRINTER pr ON p.modelno = pr.modelno

WHERE p.maker = 'Epson';

* This query returns all unique printer **types** (laser, ink-jet, etc.) where the **maker is Epson**.

**✅ b) Find those hard disk sizes which occur in two or more PCs**

sql

CopyEdit

SELECT hd, COUNT(\*) AS occurrences

FROM PC

GROUP BY hd

HAVING COUNT(\*) >= 2;

* This finds **duplicate hard disk sizes** among PCs by grouping on hd and filtering groups that appear **2 or more times**.

**✅ c) Trigger on LAPTOP table: Minimum speed should be 120 MHz**

Assuming speed is stored in MHz (like 120.0), you can create the trigger as:

sql

CopyEdit

DELIMITER //

CREATE TRIGGER check\_min\_laptop\_speed

BEFORE INSERT ON LAPTOP

FOR EACH ROW

BEGIN

IF NEW.speed < 120 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Laptop speed must be at least 120 MHz';

END IF;

END //

DELIMITER ;

If you also want to protect updates:

sql

CopyEdit

DELIMITER //

CREATE TRIGGER check\_min\_laptop\_speed\_update

BEFORE UPDATE ON LAPTOP

FOR EACH ROW

BEGIN

IF NEW.speed < 120 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Laptop speed must be at least 120 MHz';

END IF;

END //

DELIMITER ;

**✅ d) Demonstrate the use of a cursor using the PRODUCT table**

Here's a simple stored procedure using a **cursor** to fetch all product rows one by one and print them:

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE ShowAllProducts()

BEGIN

DECLARE v\_maker VARCHAR(50);

DECLARE v\_model INT;

DECLARE v\_type VARCHAR(10);

DECLARE done INT DEFAULT FALSE;

DECLARE cur CURSOR FOR

SELECT maker, modelno, type FROM PRODUCT;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_maker, v\_model, v\_type;

IF done THEN

LEAVE read\_loop;

END IF;

-- Simulate output (in MySQL use SELECT to output each row)

SELECT v\_maker AS Maker, v\_model AS ModelNo, v\_type AS Type;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

To run the cursor:

sql

CopyEdit

CALL ShowAllProducts();

**a) Find PC models having a speed of at least 150 MHz**

sql

CopyEdit

SELECT modelno, speed

FROM PC

WHERE speed >= 150;

This query returns all **PC models** where speed is **greater than or equal to 150 MHz**.

**✅ b) Find those manufacturers that sell Laptops but not PCs**

sql

CopyEdit

SELECT DISTINCT p1.maker

FROM PRODUCT p1

WHERE p1.type = 'Laptop'

AND p1.maker NOT IN (

SELECT DISTINCT p2.maker

FROM PRODUCT p2

WHERE p2.type = 'PC'

);

* This returns manufacturers who are listed as making **Laptops only**, and not **PCs**.

**✅ c) Trigger on LAPTOP table to ensure price is not less than ₹30,000**

sql

CopyEdit

DELIMITER //

CREATE TRIGGER check\_laptop\_price

BEFORE INSERT ON LAPTOP

FOR EACH ROW

BEGIN

IF NEW.price < 30000 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Laptop price must be at least ₹30,000';

END IF;

END //

DELIMITER ;

💡 *You can replicate this for BEFORE UPDATE if needed.*

**✅ d) Procedure to find the manufacturer of the most expensive laptop**

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE MostExpensiveLaptopMaker()

BEGIN

DECLARE max\_price DECIMAL(10,2);

DECLARE model INT;

SELECT MAX(price) INTO max\_price FROM LAPTOP;

SELECT modelno INTO model FROM LAPTOP WHERE price = max\_price LIMIT 1;

SELECT maker

FROM PRODUCT

WHERE modelno = model;

END //

DELIMITER ;

CREATE TABLE DOCTOR (

Did INT PRIMARY KEY,

Dname VARCHAR(100) NOT NULL,

Daddress VARCHAR(255) NOT NULL,

qualification VARCHAR(20) CHECK (qualification IN ('M.B.B.S.', 'B.A.M.S.', 'M.S.')) NOT NULL

);

CREATE TABLE PATIENTMASTER (

Pcode INT PRIMARY KEY,

Pname VARCHAR(100) NOT NULL,

Padd VARCHAR(255) NOT NULL,

age INT NOT NULL,

gender CHAR(1) CHECK (gender IN ('M', 'F')) NOT NULL,

bloodgroup VARCHAR(5) NOT NULL,

Did INT NOT NULL,

FOREIGN KEY (Did) REFERENCES DOCTOR(Did)

);

CREATE TABLE ADMITTEDPATIENT (

Pcode INT PRIMARY KEY,

entry\_date DATE NOT NULL,

discharge\_date DATE,

wardno INT CHECK (wardno < 6) NOT NULL,

disease VARCHAR(100) NOT NULL,

FOREIGN KEY (Pcode) REFERENCES PATIENTMASTER(Pcode)

);

**a) Find details of patients admitted between 03/03/2008 and 25/03/2008**

sql

CopyEdit

SELECT \*

FROM ADMITTEDPATIENT

WHERE entry\_date BETWEEN '2008-03-03' AND '2008-03-25';

**b) Find names of doctors treating patients from ‘Jalgaon’**

sql

CopyEdit

SELECT DISTINCT d.Dname

FROM DOCTOR d

JOIN PATIENTMASTER p ON d.Did = p.Did

WHERE p.Padd LIKE '%Jalgaon%';

**⚙️ 4. Procedure to Calculate Bill of Currently Admitted Patients**

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE CalculateCurrentBills()

BEGIN

SELECT

a.Pcode,

p.Pname,

DATEDIFF(CURDATE(), a.entry\_date) AS days\_admitted,

DATEDIFF(CURDATE(), a.entry\_date) \* 500 AS bill

FROM ADMITTEDPATIENT a

JOIN PATIENTMASTER p ON a.Pcode = p.Pcode

WHERE a.discharge\_date IS NULL;

END //

DELIMITER ;

-- Run using:

CALL CalculateCurrentBills();

**🚨 5. Trigger on DOCTOR: qualification validation**

This trigger prevents inserting an invalid qualification (if CHECK constraint isn't enough or not supported).

sql

CopyEdit

DELIMITER //

CREATE TRIGGER validate\_qualification

BEFORE INSERT ON DOCTOR

FOR EACH ROW

BEGIN

IF NEW.qualification NOT IN ('M.B.B.S.', 'B.A.M.S.', 'M.S.') THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Qualification must be M.B.B.S., B.A.M.S., or M.S.';

END IF;

END //

DELIMITER ;

**a) Find details of the patients who are treated by M.B.B.S. doctors**

sql

CopyEdit

SELECT p.\*

FROM PATIENTMASTER p

JOIN DOCTOR d ON p.Did = d.Did

WHERE d.qualification = 'M.B.B.S.';

**✅ b) Find name of the doctor treating male patients suffering from brain tumor and age < 40**

sql

CopyEdit

SELECT DISTINCT d.Dname

FROM DOCTOR d

JOIN PATIENTMASTER p ON d.Did = p.Did

JOIN ADMITTEDPATIENT a ON p.Pcode = a.Pcode

WHERE p.gender = 'M'

AND p.age < 40

AND a.disease = 'brain tumor';

**✅ c) Procedure to calculate bill of patients discharged on '30-03-2008'**

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE CalculateDischargedBills()

BEGIN

SELECT

a.Pcode,

p.Pname,

DATEDIFF(a.discharge\_date, a.entry\_date) AS no\_of\_days,

DATEDIFF(a.discharge\_date, a.entry\_date) \* 500 AS bill

FROM ADMITTEDPATIENT a

JOIN PATIENTMASTER p ON a.Pcode = p.Pcode

WHERE a.discharge\_date = '2008-03-30';

END //

DELIMITER ;

-- Execute:

CALL CalculateDischargedBills();

**✅ d) Cursor on DOCTOR table to fetch the first row & display the row count**

sql

CopyEdit

DELIMITER //

CREATE PROCEDURE FetchFirstDoctorAndCount()

BEGIN

DECLARE v\_did INT;

DECLARE v\_name VARCHAR(100);

DECLARE v\_address VARCHAR(255);

DECLARE v\_qualification VARCHAR(20);

DECLARE row\_count INT DEFAULT 0;

DECLARE done INT DEFAULT FALSE;

-- Cursor to fetch all rows

DECLARE cur CURSOR FOR SELECT Did, Dname, Daddress, qualification FROM DOCTOR;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

-- Count total rows

SELECT COUNT(\*) INTO row\_count FROM DOCTOR;

-- Open and fetch first row

OPEN cur;

FETCH cur INTO v\_did, v\_name, v\_address, v\_qualification;

-- Output

SELECT v\_did AS Did, v\_name AS Dname, v\_address AS Daddress, v\_qualification AS Qualification;

SELECT row\_count AS Total\_Rows;

CLOSE cur;

END //

DELIMITER ;

-- Execute:

CALL FetchFirstDoctorAndCount();