CMR Institute of Technology
Information Science and Engineering
SQL Hackathon 2025
SET 1 Solution: Climate Monitoring System
Date: 11 November 2025 | Duration: 3 Hours | Team Size: 2

Challenge 1 (Basic/Medium Level) – 50 Marks:

Objective:
Design schema, create tables, insert data, and perform basic SQL queries.

Business Scenario:
Global Climate Watch is an organization monitoring climate data across various weather stations worldwide. They need a database system to track weather stations, climate observations, environmental alerts, and research teams. You are appointed as a database developer to design and implement their climate monitoring system.

Tasks:

1. Schema Diagram (10 Marks) – Show entities and relationships:
- WeatherStation(StationID, StationName, Location, Country, Altitude)
- ResearchTeam(TeamID, TeamName, LeadScientist, StationID)
- ClimateData(DataID, StationID, RecordDate, Temperature, Humidity, Rainfall, CO2Level)
- Alert(AlertID, StationID, AlertType, AlertDate, SeverityLevel)

2. Table Creation (5 Marks) – Define primary keys, foreign keys, and constraints.

```
-- WeatherStation table
CREATE TABLE WeatherStation (
  StationID INT PRIMARY KEY AUTO_INCREMENT,
  StationName VARCHAR(150) NOT NULL,
  Location VARCHAR(150),
  Country VARCHAR(100),
  Altitude INT CHECK (Altitude >= -500));

-- ResearchTeam table
CREATE TABLE ResearchTeam (
  TeamID INT PRIMARY KEY AUTO_INCREMENT,
  TeamName VARCHAR(150) NOT NULL,
  LeadScientist VARCHAR(150),
  StationID INT,
  FOREIGN KEY (StationID) REFERENCES WeatherStation(StationID)
    ON DELETE SET NULL ON UPDATE CASCADE);

-- ClimateData table
CREATE TABLE ClimateData (
  DataID INT PRIMARY KEY AUTO_INCREMENT,
  StationID INT NOT NULL,
  RecordDate DATE NOT NULL,
  Temperature DECIMAL(5,2), -- Celsius
  Humidity DECIMAL(5,2), -- percentage
  Rainfall DECIMAL(6,2), -- mm
  CO2Level DECIMAL(7,2), -- ppm
  FOREIGN KEY (StationID) REFERENCES WeatherStation(StationID)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

```
-- Alert table
CREATE TABLE Alert (
  AlertID INT PRIMARY KEY AUTO_INCREMENT,
  StationID INT NOT NULL,
  AlertType VARCHAR(100),
  AlertDate DATETIME DEFAULT CURRENT_TIMESTAMP,
  SeverityLevel VARCHAR(20), -- e.g., Low, Medium, High, Critical
  FOREIGN KEY (StationID) REFERENCES WeatherStation(StationID)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

3. Insert Data (5 Marks) – Insert at least 5 records per table.

```
-- WeatherStation inserts
INSERT INTO WeatherStation(StationName, Location, Country, Altitude) VALUES
('Mount Aurora', 'Aurora Peak', 'Finland', 1350),
('Lakeview Station', 'Lake District', 'Iceland', 50),
('Desert Edge', 'Rajasthan Desert', 'India', 320),
('High Ridge', 'Andes Ridge', 'Peru', 2200),
('Valley Base', 'Green Valley', 'USA', 200);

-- ResearchTeam inserts
INSERT INTO ResearchTeam(TeamName, LeadScientist, StationID) VALUES
('Aurora Team', 'Dr. Anil Kumar', 1),
('Lake Monitors', 'Dr. Sita Sharma', 2),
('Desert Study Group', 'Prof. Ramesh Kumar', 3),
('Andes Research', 'Dr. Lucia Vega', 4),
('Valley Ecology', 'Dr. John Doe', 5);

-- ClimateData inserts (sample dates)
INSERT INTO ClimateData(StationID, RecordDate, Temperature, Humidity, Rainfall, CO2Level)
VALUES
(1,'2025-11-01', -5.4, 78.5, 0.0, 395.20),
(1,'2025-11-02', -3.1, 80.2, 0.0, 396.00),
(2,'2025-11-01', 4.0, 85.0, 2.5, 397.50),
(3,'2025-11-01', 38.5, 20.0, 0.0, 405.60),
(4,'2025-11-01', 10.3, 60.0, 5.0, 410.10),
(5,'2025-11-01', 25.0, 55.0, 12.0, 402.00);

-- Alert inserts
INSERT INTO Alert(StationID, AlertType, AlertDate, SeverityLevel) VALUES
(3, 'HeatWave', '2025-06-12 10:00:00', 'High'),
(1, 'SnowStorm', '2025-01-20 02:30:00', 'Medium'),
(2, 'FloodRisk', '2025-09-15 08:00:00', 'High'),
(4, 'LandslideRisk', '2025-03-05 12:00:00', 'Critical'),
(5, 'WaterLogging', '2025-07-24 18:00:00', 'Low');
```

4. SQL Queries (20 Marks) – Write queries for:

- Weather stations with altitude > 1000 meters

```
SELECT * FROM WeatherStation
WHERE Altitude > 1000;
```

- Total rainfall recorded per station

```
SELECT w.StationID, w.StationName, COALESCE(SUM(c.Rainfall),0) AS TotalRainfall
FROM WeatherStation w
```

```
LEFT JOIN ClimateData c ON w.StationID = c.StationID
GROUP BY w.StationID, w.StationName;
```

- Stations in same country as 'USA'

```
SELECT * FROM WeatherStation
WHERE Country = 'USA';
SELECT w2.*
FROM WeatherStation w1
JOIN WeatherStation w2 ON w1.Country = w2.Country
WHERE w1.StationName = 'Valley Base';
```

- Climate data with temperature > 35°C and CO2Level > 400 ppm

```
SELECT * FROM ClimateData
WHERE Temperature > 35 AND CO2Level > 400;
```

- Count of alerts per severity level

```
SELECT SeverityLevel, COUNT(*) AS AlertCount
FROM Alert
GROUP BY SeverityLevel
ORDER BY AlertCount DESC;
```

5.Pattern matching Queries (10 Marks)

-Find all weather stations whose name starts with 'Mount'

```
SELECT * FROM WeatherStation
WHERE StationName LIKE 'Mount%';
```

-Find all stations located in countries ending with 'land' (e.g., Finland, Iceland)

```
SELECT * FROM WeatherStation
WHERE Country LIKE '%land';
```

-Find all lead scientists whose name contains 'Kumar' anywhere

```
SELECT * FROM ResearchTeam
WHERE LeadScientist LIKE '%Kumar%';
```

-Find all alerts with AlertType starting with 'Heat' (e.g., HeatWave, HeatStress)

```
SELECT * FROM Alert
WHERE AlertType LIKE 'Heat%';
```

-Find stations whose location has exactly 5 characters ending with 'hi' (e.g., Delhi)

```
SELECT * FROM WeatherStation
WHERE Location LIKE '___hi' -- 5 chars total, last two 'hi'
  AND LENGTH(Location) = 5;
```

Challenge 2 (Advanced Level) – 50 Marks

Objective: Work with joins, views, and triggers to ensure data integrity and analytical reporting.

Tasks:

1. Joins (10 Marks)– Queries using joins:

- Station Name, Country, Team Name, Lead Scientist

```
SELECT w.StationName, w.Country, r.TeamName, r.LeadScientist
FROM WeatherStation w
LEFT JOIN ResearchTeam r ON w.StationID = r.StationID;
```

- Stations with no alerts recorded

```
SELECT w.StationID, w.StationName
FROM WeatherStation w
LEFT JOIN Alert a ON w.StationID = a.StationID
WHERE a.AlertID IS NULL;
```

2. Views (10 Marks) – Create:

- HighRiskStationsView(StationName, Location, AlertCount, AvgTemperature)

```
CREATE OR REPLACE VIEW HighRiskStationsView AS
SELECT w.StationName, w.Location,
      COUNT(a.AlertID) AS AlertCount,
      ROUND(AVG(c.Temperature),2) AS AvgTemperature
FROM WeatherStation w
LEFT JOIN Alert a ON w.StationID = a.StationID
LEFT JOIN ClimateData c ON w.StationID = c.StationID
GROUP BY w.StationID, w.StationName, w.Location
HAVING COUNT(a.AlertID) > 0; -- optionally filter to stations with alerts
```

- CountrySummaryView(Country, TotalStations, AvgTemperature, TotalRainfall)

```
CREATE OR REPLACE VIEW CountrySummaryView AS
SELECT w.Country,
      COUNT(DISTINCT w.StationID) AS TotalStations,
      ROUND(AVG(c.Temperature),2) AS AvgTemperature,
      ROUND(SUM(c.Rainfall),2) AS TotalRainfall
FROM WeatherStation w
LEFT JOIN ClimateData c ON w.StationID = c.StationID
GROUP BY w.Country;
```

3. Triggers (15 Marks) – Create trigger on new alert insert:

- AlertLog(AlertID, StationID, LoggedDate, Action)

```
-- Create the log table
CREATE TABLE AlertLog (
  LogID INT PRIMARY KEY AUTO_INCREMENT,
  AlertID INT,
  StationID INT,
  LoggedDate DATETIME DEFAULT CURRENT_TIMESTAMP,
  Action VARCHAR(100)
);

-- Trigger to insert into AlertLog after insert on Alert (MySQL style)
DELIMITER $$
CREATE TRIGGER trg_after_alert_insert
AFTER INSERT ON Alert
FOR EACH ROW
BEGIN
  INSERT INTO AlertLog(AlertID, StationID, LoggedDate, Action)
  VALUES (NEW.AlertID, NEW.StationID, NOW(), CONCAT('Alert inserted: ', NEW.AlertType));
END$$
DELIMITER ;
```

4. Analytical Queries (15 Marks):

- Top 2 countries with the highest average temperature

```
SELECT w.Country, AVG(c.Temperature) AS AvgTemp
FROM WeatherStation w
```

```
JOIN ClimateData c ON w.StationID = c.StationID
GROUP BY w.Country
ORDER BY AvgTemp DESC
LIMIT 2;
```

- Average humidity by location (group by country)

```
SELECT w.Country, w.Location, ROUND(AVG(c.Humidity),2) AS AvgHumidity
FROM WeatherStation w
JOIN ClimateData c ON w.StationID = c.StationID
GROUP BY w.Country, w.Location;
```

- Stations where average CO2 level exceeds 450 ppm

```
SELECT w.StationID, w.StationName, ROUND(AVG(c.CO2Level),2) AS AvgCO2
FROM WeatherStation w
JOIN ClimateData c ON w.StationID = c.StationID
GROUP BY w.StationID, w.StationName
HAVING AVG(c.CO2Level) > 450;
```

CMR Institute of Technology
Information Science and Engineering
SQL Hackathon 2025
SET 2 SOLUTION: Employee Management System
Date: 11 November 2025 | Duration: 3 Hours | Team Size: 2

Challenge 1 (Basic/Medium Level) – 50 Marks

Objective:
Design schema, create tables, insert data, and perform basic SQL queries.

Business Scenario:
TechCorp Solutions is a growing IT company that needs a comprehensive employee management system. The company wants to track employees, their departments, projects, and attendance records. You are hired as a database developer to design and implement their HR management system.

Tasks:

1. Schema Diagram (10 Marks)– Show entities and relationships:

- Department(DeptID, DeptName, Location, ManagerID)
- Employee(EmpID, Name, Gender, Phone, Email, Salary, JoinDate, DeptID)
- Project(ProjectID, ProjectName, StartDate, Budget, DeptID)
- Attendance(AttendanceID, EmpID, AttendanceDate, Status, WorkingHours)

2. Table Creation (5 Marks)– Define primary keys, foreign keys, and constraints.

```
CREATE TABLE Department (
  DeptID INT PRIMARY KEY AUTO_INCREMENT,
  DeptName VARCHAR(100) NOT NULL,
  Location VARCHAR(100),
  ManagerID INT NULL);

CREATE TABLE Employee (
  EmpID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(150) NOT NULL,
  Gender VARCHAR(10),
  Phone VARCHAR(15),
  Email VARCHAR(150) UNIQUE,
  Salary DECIMAL(12,2),
  JoinDate DATE,
  DeptID INT,
  FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
    ON DELETE SET NULL ON UPDATE CASCADE);

-- After Employee table exists, set Department.ManagerID FK (optional)
ALTER TABLE Department
  ADD CONSTRAINT fk_dept_manager
  FOREIGN KEY (ManagerID) REFERENCES Employee(EmpID)
  ON DELETE SET NULL ON UPDATE CASCADE;

CREATE TABLE Project (
```

```
  ProjectID INT PRIMARY KEY AUTO_INCREMENT,
  ProjectName VARCHAR(150) NOT NULL,
  StartDate DATE,
  Budget DECIMAL(15,2),
  DeptID INT,
  FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
    ON DELETE SET NULL ON UPDATE CASCADE);

CREATE TABLE Attendance (
  AttendanceID INT PRIMARY KEY AUTO_INCREMENT,
  EmpID INT NOT NULL,
  AttendanceDate DATE NOT NULL,
  Status VARCHAR(20), -- 'Present','Absent','Leave'
  WorkingHours DECIMAL(4,2),
  FOREIGN KEY (EmpID) REFERENCES Employee(EmpID)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

3. Insert Data (5 Marks) – Insert at least 5 records per table.

```
-- Departments
INSERT INTO Department(DeptName, Location) VALUES
('Engineering', 'Bengaluru'),
('HR', 'Bengaluru'),
('Research', 'Chennai'),
('Sales', 'Mumbai'),
('Support', 'Bengaluru');

-- Employees
INSERT INTO Employee(Name, Gender, Phone, Email, Salary, JoinDate, DeptID) VALUES
('Anita Rao','F','9898989898','anita.rao@gmail.com',60000,'2022-03-15',1),
('Suresh Kumar','M','9876543210','suresh.kumar@gmail.com',45000,'2021-07-12',1),
('Asha Nair','F','9812345670','asha.nair@gmail.com',55000,'2020-11-01',2),
('Ravi Singh','M','9900112233','ravi.singh@yahoo.com',52000,'2019-02-20',3),
('Samir Patel','M','9810099887','samir.patel@gmail.com',70000,'2018-05-10',4);

-- Update Department.ManagerID for demonstration
UPDATE Department SET ManagerID = 1 WHERE DeptID = 1;
UPDATE Department SET ManagerID = 3 WHERE DeptID = 2;

-- Projects
INSERT INTO Project(ProjectName, StartDate, Budget, DeptID) VALUES
('AI-Healthcare','2024-02-01',200000,'1'),
('Cloud Migration','2024-09-10',150000,'1'),
('HR-Portal','2025-01-15',80000,'2'),
('Sales-CRM','2024-05-20',120000,'4'),
('AI-Finance','2025-04-01',220000,'1');
```

```
-- Attendance (sample)
INSERT INTO Attendance(EmpID, AttendanceDate, Status, WorkingHours) VALUES
(1,'2025-11-01','Present',8.0),
(2,'2025-11-01','Present',8.0),
(3,'2025-11-01','Absent',0.0),
(4,'2025-11-01','Present',7.5),
(5,'2025-11-01','Present',8.0);
```

4. SQL Queries (20 Marks) – Write queries for:

- Employees earning salary > 50000

```
SELECT * FROM Employee WHERE Salary > 50000;
```

- Total working hours per employee

```
SELECT e.EmpID, e.Name, COALESCE(SUM(a.WorkingHours),0) AS TotalHours
FROM Employee e
LEFT JOIN Attendance a ON e.EmpID = a.EmpID
GROUP BY e.EmpID, e.Name;
```

- Employees in same location as their department

```
-- Assuming Employee has no location; using Department.Location and Employee's DeptID
SELECT e.EmpID, e.Name, d.DeptName, d.Location
FROM Employee e
JOIN Department d ON e.DeptID = d.DeptID
WHERE d.Location = d.Location; -- trivial; if employees had Location column, compare
-- If employee had Location column use: WHERE e.Location = d.Location
```

- Projects with budget > 100000 and start date after '2024-01-01'

```
SELECT * FROM Project
WHERE Budget > 100000 AND StartDate > '2024-01-01';
```

- Count of employees per department

```
SELECT d.DeptName, COUNT(e.EmpID) AS EmployeeCount
FROM Department d
LEFT JOIN Employee e ON d.DeptID = e.DeptID
GROUP BY d.DeptName;
```

5.Pattern matching Queries (10 Marks)

-Find all employees whose name starts with 'A' or 'S'

```
SELECT * FROM Employee
WHERE Name LIKE 'A%' OR Name LIKE 'S%';
```

-Find all employees with email addresses from 'gmail.com' domain

```
SELECT * FROM Employee
WHERE Email LIKE '%@gmail.com';
```

-Find all departments whose name contains 'Tech' anywhere

```
SELECT * FROM Department
WHERE DeptName LIKE '%Tech%';
```

-Find all projects starting with 'AI' (e.g., AI-Healthcare, AI-Finance)

```
SELECT * FROM Project
WHERE ProjectName LIKE 'AI%';
```

-Find employees whose phone number starts with '98' (10-digit format)

```
SELECT * FROM Employee
WHERE Phone LIKE '98%';
```

Challenge 2 (Advanced Level) – 50 Marks

Objective: Work with joins, views, and triggers to ensure data integrity and analytical reporting.

Tasks:

1. Joins (10 Marks) – Queries using joins:

- Employee Name, Department Name, Project Name, Salary

```
SELECT e.Name AS EmployeeName, d.DeptName, p.ProjectName, e.Salary
FROM Employee e
LEFT JOIN Department d ON e.DeptID = d.DeptID
LEFT JOIN Project p ON d.DeptID = p.DeptID;
```

- Employees who have never been absent (Status = 'Present' for all records)

```
SELECT e.EmpID, e.Name
FROM Employee e
LEFT JOIN Attendance a ON e.EmpID = a.EmpID
GROUP BY e.EmpID, e.Name
HAVING SUM(CASE WHEN a.Status = 'Absent' THEN 1 ELSE 0 END) = 0;
```

2. Views (10 Marks) – Create:

- DepartmentSummaryView(DeptName, TotalEmployees, AvgSalary, TotalBudget)

```
CREATE OR REPLACE VIEW DepartmentSummaryView AS
SELECT d.DeptName,
      COUNT(e.EmpID) AS TotalEmployees,
      ROUND(AVG(e.Salary),2) AS AvgSalary,
      ROUND(COALESCE(SUM(p.Budget),0),2) AS TotalBudget
FROM Department d
LEFT JOIN Employee e ON d.DeptID = e.DeptID
LEFT JOIN Project p ON d.DeptID = p.DeptID
GROUP BY d.DeptName;
```

- EmployeePerformanceView(EmpName, TotalWorkingHours, AttendanceRate, Salary)

```
CREATE OR REPLACE VIEW EmployeePerformanceView AS
SELECT e.Name AS EmpName,
      COALESCE(SUM(a.WorkingHours),0) AS TotalWorkingHours,
      ROUND(100.0 * SUM(CASE WHEN a.Status = 'Present' THEN 1 ELSE 0 END) /
NULLIF(COUNT(a.AttendanceID),0),2) AS AttendanceRate,
      e.Salary
FROM Employee e
LEFT JOIN Attendance a ON e.EmpID = a.EmpID
GROUP BY e.EmpID, e.Name, e.Salary;
```

3. Triggers (15 Marks) – Create trigger on new employee insert:
- EmployeeAudit(EmpID, DeptID, HireDate, Action)

```
CREATE TABLE EmployeeAudit (
  AuditID INT PRIMARY KEY AUTO_INCREMENT,
  EmpID INT,
  DeptID INT,
  HireDate DATE,
  Action VARCHAR(100),
  AuditDate DATETIME DEFAULT CURRENT_TIMESTAMP
);

DELIMITER $$
CREATE TRIGGER trg_after_employee_insert
AFTER INSERT ON Employee
FOR EACH ROW
BEGIN
  INSERT INTO EmployeeAudit(EmpID, DeptID, HireDate, Action)
  VALUES (NEW.EmpID, NEW.DeptID, NEW.JoinDate, 'Employee Inserted');
END$$
DELIMITER ;
```

4. Analytical Queries (15 Marks):
- Top 2 departments by total salary expenditure

```
SELECT d.DeptName, ROUND(SUM(e.Salary),2) AS TotalSalary
FROM Department d
JOIN Employee e ON d.DeptID = e.DeptID
GROUP BY d.DeptName
ORDER BY TotalSalary DESC
LIMIT 2;
```

- Average working hours by department

```
SELECT d.DeptName, ROUND(AVG(a.WorkingHours),2) AS AvgWorkingHours
FROM Department d
JOIN Employee e ON d.DeptID = e.DeptID
JOIN Attendance a ON e.EmpID = a.EmpID
GROUP BY d.DeptName;
```

- Employees with attendance rate < 80%

```
SELECT p.EmpName, p.AttendanceRate
FROM EmployeePerformanceView p
WHERE p.AttendanceRate < 80;
```

CMR Institute of Technology
Information Science and Engineering
SQL Hackathon 2025
SET 3 SOLUTION: Library Management System
Date: 11 November 2025 | Duration: 3 Hours | Team Size: 2

Challenge 1 (Basic/Medium Level) – 50 Marks

Objective:
Design schema, create tables, insert data, and perform basic SQL queries.

Business Scenario:
City Central Library is modernizing its operations and needs a digital database system to manage books, members, book issues, and library branches. You are appointed as a database developer to design and implement their library management system.

Tasks:

1. Schema Diagram (10 Marks) – Show entities and relationships:
- Branch(BranchID, BranchName, Location, ContactNumber)
- Member(MemberID, Name, Gender, Phone, Email, MembershipType, JoinDate, BranchID)
- Book(BookID, Title, Author, Genre, Publisher, Price, StockQuantity, BranchID)
- Issue(IssueID, MemberID, BookID, IssueDate, ReturnDate, Fine)

2. Table Creation (10 Marks) – Define primary keys, foreign keys, and constraints.

```
CREATE TABLE Branch (
  BranchID INT PRIMARY KEY AUTO_INCREMENT,
  BranchName VARCHAR(150) NOT NULL,
  Location VARCHAR(150),
  ContactNumber VARCHAR(20));

CREATE TABLE Member (
  MemberID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(150) NOT NULL,
  Gender VARCHAR(10),
  Phone VARCHAR(15),
  Email VARCHAR(150) UNIQUE,
  MembershipType VARCHAR(50), -- e.g., Regular, Premium
  JoinDate DATE,
  BranchID INT,
  FOREIGN KEY (BranchID) REFERENCES Branch(BranchID)
    ON DELETE SET NULL ON UPDATE CASCADE);

CREATE TABLE Book (
  BookID INT PRIMARY KEY AUTO_INCREMENT,
  Title VARCHAR(250) NOT NULL,
  Author VARCHAR(150),
  Genre VARCHAR(100),
  Publisher VARCHAR(150),
  Price DECIMAL(10,2),
```

```
  StockQuantity INT,
  BranchID INT,
  FOREIGN KEY (BranchID) REFERENCES Branch(BranchID)
    ON DELETE SET NULL ON UPDATE CASCADE);

CREATE TABLE Issue (
  IssueID INT PRIMARY KEY AUTO_INCREMENT,
  MemberID INT NOT NULL,
  BookID INT NOT NULL,
  IssueDate DATE,
  ReturnDate DATE,
  Fine DECIMAL(8,2) DEFAULT 0.00,
  FOREIGN KEY (MemberID) REFERENCES Member(MemberID)
    ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (BookID) REFERENCES Book(BookID)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

3. Insert Data (10 Marks) – Insert at least 5 records per table.

```
-- Branches
INSERT INTO Branch(BranchName, Location, ContactNumber) VALUES
('Central Library','Central Avenue','080-1234567'),
('North Branch','North Central','080-7654321'),
('South Branch','South Park','080-1112223'),
('East Branch','East End','080-3334445'),
('West Branch','West Lane','080-5556667');

-- Members
INSERT INTO Member(Name, Gender, Phone, Email, MembershipType, JoinDate, BranchID)
VALUES
('Rahul Sharma','M','9876501234','rahul.sharma@edu.com','Premium','2024-01-15',1),
('Priya Nair','F','9812345678','priya.nair@gmail.com','Regular','2023-03-10',1),
('Amit Kumar','M','9898989890','amit.kumar@college.edu','Premium','2022-06-05',2),
('Sonia Gupta','F','9800112233','sonia.gupta@edu.com','Regular','2021-09-12',3),
('Neha Rao','F','9770011223','neha.rao@domain.com','Premium','2025-02-20',4);

-- Books
INSERT INTO Book(Title, Author, Genre, Publisher, Price, StockQuantity, BranchID) VALUES
('The Lost Kingdom','R. Sharma','Fiction','PenPress',399.00,3,1),
('Mystery of Rivers','A. Roy','Mystery','PenWorld',499.00,2,1),
('Data Structures','S. Kumar','Education','AcademicPub',599.00,5,2),
('Modern Physics','P. Singh','Academic','SciBooks',799.00,1,3),
('Gardening 101','L. Green','Hobby','HomePress',299.00,4,4);

-- Issues
INSERT INTO Issue(MemberID, BookID, IssueDate, ReturnDate, Fine) VALUES
(1,1,'2025-10-01','2025-10-21',0.00),
```

```
(2,2,'2025-10-05',NULL,0.00),
(3,3,'2025-09-20','2025-10-05',0.00),
(4,4,'2025-09-25','2025-10-10',10.00),
(5,5,'2025-10-10',NULL,0.00);
```

4. SQL Queries (20 Marks) – Write queries for:

- Books of genre 'Fiction' or 'Mystery'

```
SELECT * FROM Book
WHERE Genre IN ('Fiction','Mystery');
```

- Total books issued per member

```
SELECT m.MemberID, m.Name, COUNT(i.IssueID) AS BooksIssued
FROM Member m
LEFT JOIN Issue i ON m.MemberID = i.MemberID
GROUP BY m.MemberID, m.Name;
```

- Members with 'Premium' membership type

```
SELECT * FROM Member WHERE MembershipType = 'Premium';
```

- Books with price > 500 and stock quantity < 5

```
SELECT * FROM Book
WHERE Price > 500 AND StockQuantity < 5;
```

- Total fine collected per branch

```
SELECT b.BranchName, COALESCE(SUM(i.Fine),0) AS TotalFine
FROM Branch b
LEFT JOIN Member m ON b.BranchID = m.BranchID
LEFT JOIN Issue i ON m.MemberID = i.MemberID
GROUP BY b.BranchName;
```

5.Pattern matching Queries (10 Marks)

-Find all books whose title contains 'The' anywhere

```
SELECT * FROM Book WHERE Title LIKE '%The%';
```

-Find all authors whose last name is 'Sharma' (assuming format: FirstName Sharma)

```
SELECT * FROM Book WHERE Author LIKE '% Sharma';
```

-Find all members with email ending with '.edu' (educational domain)

```
SELECT * FROM Member WHERE Email LIKE '%.edu';
```

-Find all books published by publishers starting with 'Pen' (e.g., Penguin, Penton)

```
SELECT * FROM Book WHERE Publisher LIKE 'Pen%';
```

-Find all branches located in areas containing 'Central' (e.g., Central Avenue, North Central)

```
SELECT * FROM Branch WHERE Location LIKE '%Central%';
```

Challenge 2 (Advanced Level) – 50 Marks

Objective: Work with joins, views, and triggers to ensure data integrity and analytical reporting.

Tasks:

1. Joins (10 Marks) – Queries using joins:

- Member Name, Book Title, Branch Name, Issue Date

```
SELECT m.Name AS MemberName, b.Title AS BookTitle, br.BranchName, i.IssueDate
FROM Issue i
JOIN Member m ON i.MemberID = m.MemberID
JOIN Book b ON i.BookID = b.BookID
JOIN Branch br ON m.BranchID = br.BranchID;
```

- Members who have never issued any book

```
SELECT m.MemberID, m.Name
FROM Member m
LEFT JOIN Issue i ON m.MemberID = i.MemberID
WHERE i.IssueID IS NULL;
```

2. Views (10 Marks) – Create:

- ActiveIssuesView(MemberName, BookTitle, IssueDate, DaysOverdue)

```
CREATE OR REPLACE VIEW ActiveIssuesView AS
SELECT m.Name AS MemberName, b.Title AS BookTitle, i.IssueDate,
    CASE
      WHEN i.ReturnDate IS NULL THEN DATEDIFF(CURDATE(), i.IssueDate)
      ELSE GREATEST(0, DATEDIFF(i.ReturnDate, i.IssueDate))
    END AS DaysOverdue
FROM Issue i
JOIN Member m ON i.MemberID = m.MemberID
JOIN Book b ON i.BookID = b.BookID;
```

- BranchInventoryView(BranchName, TotalBooks, TotalMembers, TotalFines)

```
CREATE OR REPLACE VIEW BranchInventoryView AS
SELECT br.BranchName,
    COALESCE(SUM(b.StockQuantity),0) AS TotalBooks,
    COALESCE(COUNT(DISTINCT m.MemberID),0) AS TotalMembers,
    COALESCE(SUM(i.Fine),0) AS TotalFines
FROM Branch br
LEFT JOIN Book b ON br.BranchID = b.BranchID
LEFT JOIN Member m ON br.BranchID = m.BranchID
LEFT JOIN Issue i ON m.MemberID = i.MemberID
GROUP BY br.BranchName;
```

3. Triggers (15 Marks) – Create trigger on book issue insert:

- IssueLog(IssueID, MemberID, BookID, LogDate, Action)

```
CREATE TABLE IssueLog (
  LogID INT PRIMARY KEY AUTO_INCREMENT,
```

```
  IssueID INT,
  MemberID INT,
  BookID INT,
  LogDate DATETIME DEFAULT CURRENT_TIMESTAMP,
  Action VARCHAR(100)
);

DELIMITER $$
CREATE TRIGGER trg_after_issue_insert
AFTER INSERT ON Issue
FOR EACH ROW
BEGIN
  INSERT INTO IssueLog(IssueID, MemberID, BookID, Action)
  VALUES (NEW.IssueID, NEW.MemberID, NEW.BookID, 'Book Issued');
  -- Update stock quantity
  UPDATE Book SET StockQuantity = GREATEST(0, StockQuantity - 1) WHERE BookID = NEW.BookID;
END$$
DELIMITER ;
```

4. Analytical Queries (15 Marks):

- Top 2 branches by total books in stock

```
SELECT br.BranchName, SUM(b.StockQuantity) AS TotalStock
FROM Branch br
JOIN Book b ON br.BranchID = b.BranchID
GROUP BY br.BranchName
ORDER BY TotalStock DESC
LIMIT 2;
```

- Average fine amount by membership type

```
SELECT m.MembershipType, ROUND(AVG(i.Fine),2) AS AvgFine
FROM Member m
JOIN Issue i ON m.MemberID = i.MemberID
GROUP BY m.MembershipType;
```

- Most issued books (books issued more than 10 times)

```
SELECT b.BookID, b.Title, COUNT(i.IssueID) AS IssueCount
FROM Book b
JOIN Issue i ON b.BookID = i.BookID
GROUP BY b.BookID, b.Title
HAVING COUNT(i.IssueID) > 10
ORDER BY IssueCount DESC;
```

CMR Institute of Technology
Information Science and Engineering
SQL Hackathon 2025
SET 4 SOLUTION: Bank Management System
Date: 11 November 2025 | Duration: 3 Hours | Team Size: 2

Challenge 1 (Basic/Medium Level) – 50 Marks

Objective:
Design schema, create tables, insert data, and perform basic SQL queries.

Business Scenario:
SecureBank Ltd. is establishing a new digital banking platform and needs a robust database system to manage branches, customers, accounts, and transactions. You are appointed as a database developer to design and implement their banking management system.

Tasks:

1. Schema Diagram (10 Marks) – Show entities and relationships:
- Branch(BranchID, BranchName, City, IFSC_Code, ManagerName)
- Customer(CustomerID, Name, Gender, Phone, Email, Address, BranchID)
- Account(AccountID, CustomerID, AccountType, Balance, OpenDate, Status)
- Transaction(TransactionID, AccountID, TransactionType, Amount, TransactionDate, Description)

2. Table Creation (10 Marks) – Define primary keys, foreign keys, and constraints.

```
CREATE TABLE Branch (
  BranchID INT PRIMARY KEY AUTO_INCREMENT,
  BranchName VARCHAR(150) NOT NULL,
  City VARCHAR(100),
  IFSC_Code VARCHAR(20) UNIQUE,
  ManagerName VARCHAR(150));

CREATE TABLE Customer (
  CustomerID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(150) NOT NULL,
  Gender VARCHAR(10),
  Phone VARCHAR(15),
  Email VARCHAR(150) UNIQUE,
  Address VARCHAR(250),
  BranchID INT,
  FOREIGN KEY (BranchID) REFERENCES Branch(BranchID)
    ON DELETE SET NULL ON UPDATE CASCADE);

CREATE TABLE Account (
  AccountID INT PRIMARY KEY AUTO_INCREMENT,
  CustomerID INT NOT NULL,
  AccountType VARCHAR(50), -- e.g., Savings, Current
  Balance DECIMAL(15,2) DEFAULT 0.00,
  OpenDate DATE,
```

```
  Status VARCHAR(20) DEFAULT 'Active',
  FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
    ON DELETE CASCADE ON UPDATE CASCADE);

CREATE TABLE `Transaction` (
  TransactionID INT PRIMARY KEY AUTO_INCREMENT,
  AccountID INT NOT NULL,
  TransactionType VARCHAR(50), -- Deposit / Withdrawal / Transfer
  Amount DECIMAL(15,2),
  TransactionDate DATETIME DEFAULT CURRENT_TIMESTAMP,
  Description VARCHAR(255),
  FOREIGN KEY (AccountID) REFERENCES Account(AccountID)
    ON DELETE CASCADE ON UPDATE CASCADE);
```

3. Insert Data (10 Marks)– Insert at least 5 records per table.

```
-- Branches
INSERT INTO Branch(BranchName, City, IFSC_Code, ManagerName) VALUES
('Central Branch','Bangalore','SBIN0001234','Mr. Ramesh'),
('East Branch','Bangalore','SBIN0002345','Ms. Priya'),
('North Branch','Chennai','HDFC0009876','Mr. Karthik'),
('West Branch','Mumbai','SBIN0003456','Ms. Rekha'),
('South Branch','Hyderabad','ICIC0004567','Mr. Anand');

-- Customers
INSERT INTO Customer(Name, Gender, Phone, Email, Address, BranchID) VALUES
('Manish Kumar','M','9876501122','manish.kumar@gmail.com','12 MG Road','1'),
('Sangeeta Kumari','F','9810012345','sangeeta.kumari@mail.com','45 Park St','2'),
('Rohan Patel','M','9898981234','rohan.patel@domain.com','78 Lake Rd','1'),
('Priya Singh','F','9800112233','priya.singh@gmail.com','9 Oak St','3'),
('Kavita Kumari','F','9770011223','kavita.kumari@abc.com','33 Main St','4');

-- Accounts
INSERT INTO Account(CustomerID, AccountType, Balance, OpenDate, Status) VALUES
(1,'Savings',120000.00,'2020-05-10','Active'),
(2,'Savings',45000.00,'2022-07-20','Active'),
(3,'Current',250000.00,'2019-11-01','Active'),
(4,'Savings',80000.00,'2021-02-14','Active'),
(5,'Savings',30000.00,'2023-01-05','Active');

-- Transactions
INSERT INTO `Transaction`(AccountID, TransactionType, Amount, TransactionDate, Description)
VALUES
(1,'Deposit',50000.00,'2025-10-01 09:00:00','Salary Credit'),
(1,'Withdrawal',10000.00,'2025-10-05 11:30:00','ATM Withdrawal'),
(2,'Deposit',20000.00,'2025-09-25 10:00:00','Bank Transfer'),
(3,'Withdrawal',5000.00,'2025-10-03 15:00:00','Payment'),
```

(4,'Deposit',10000.00,'2025-10-07 12:45:00','Online Transfer');

4. SQL Queries (20 Marks) – Write queries for:

- Customers with 'Savings' account type

```
SELECT DISTINCT c.CustomerID, c.Name, a.AccountType
FROM Customer c
JOIN Account a ON c.CustomerID = a.CustomerID
WHERE a.AccountType = 'Savings';
```

- Total balance per branch

```
SELECT b.BranchName, ROUND(SUM(a.Balance),2) AS TotalBalance
FROM Branch b
LEFT JOIN Customer c ON b.BranchID = c.BranchID
LEFT JOIN Account a ON c.CustomerID = a.CustomerID
GROUP BY b.BranchName;
```

- Accounts with balance > 100000

```
SELECT * FROM Account WHERE Balance > 100000;
```

- Transactions of type 'Withdrawal' with amount > 5000

```
SELECT * FROM `Transaction`
WHERE TransactionType = 'Withdrawal' AND Amount > 5000;
```

- Count of customers per branch

```
SELECT b.BranchName, COUNT(c.CustomerID) AS CustomerCount
FROM Branch b
LEFT JOIN Customer c ON b.BranchID = c.BranchID
GROUP BY b.BranchName;
```

5.Pattern matching Queries (10 Marks)

-Find all branches whose IFSC code starts with 'SBIN' (State Bank of India)

```
SELECT * FROM Branch WHERE IFSC_Code LIKE 'SBIN%';
```

-Find all customers whose name ends with 'Kumar' or 'Kumari'

```
SELECT * FROM Customer
WHERE Name LIKE '%Kumar' OR Name LIKE '%Kumari';
```

-Find all customers living in addresses containing 'Street' or 'St'

```
SELECT * FROM Customer
WHERE Address LIKE '%Street%' OR Address LIKE '%St%';
```

-Find all transactions with description starting with 'Online' (e.g., Online Purchase, Online Transfer)

```
SELECT * FROM `Transaction`
WHERE Description LIKE 'Online%';
```

-Find all branches located in cities starting with 'B' (e.g., Bangalore, Bhopal)

```
SELECT * FROM Branch WHERE City LIKE 'B%';
```

Challenge 2 (Advanced Level) – 50 Marks

Objective: Work with joins, views, and triggers to ensure data integrity and analytical reporting.

Tasks:

1. Joins (10 Marks)– Queries using joins:

- Customer Name, Account Type, Branch Name, Current Balance

```
SELECT c.Name AS CustomerName, a.AccountType, b.BranchName, a.Balance
FROM Customer c
JOIN Account a ON c.CustomerID = a.CustomerID
LEFT JOIN Branch b ON c.BranchID = b.BranchID;
```

- Customers with no transactions in last 6 months

```
SELECT c.CustomerID, c.Name
FROM Customer c
JOIN Account a ON c.CustomerID = a.CustomerID
LEFT JOIN `Transaction` t ON a.AccountID = t.AccountID
  AND t.TransactionDate >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY c.CustomerID, c.Name
HAVING SUM(CASE WHEN t.TransactionID IS NOT NULL THEN 1 ELSE 0 END) = 0;
```

2. Views (10 Marks) – Create:

- AccountSummaryView(CustomerName, AccountType, TotalDeposits, TotalWithdrawals, CurrentBalance)

```
CREATE OR REPLACE VIEW AccountSummaryView AS
SELECT c.Name AS CustomerName,
    a.AccountType,
    COALESCE(SUM(CASE WHEN t.TransactionType = 'Deposit' THEN t.Amount ELSE 0 END),0) AS TotalDeposits,
    COALESCE(SUM(CASE WHEN t.TransactionType = 'Withdrawal' THEN t.Amount ELSE 0 END),0) AS TotalWithdrawals,
    a.Balance AS CurrentBalance
FROM Account a
JOIN Customer c ON a.CustomerID = c.CustomerID
LEFT JOIN `Transaction` t ON a.AccountID = t.AccountID
GROUP BY c.Name, a.AccountID;
```

- BranchPerformanceView(BranchName, TotalCustomers, TotalBalance, TransactionCount)

```
CREATE OR REPLACE VIEW BranchPerformanceView AS
SELECT b.BranchName,
    COUNT(DISTINCT c.CustomerID) AS TotalCustomers,
    ROUND(COALESCE(SUM(a.Balance),0),2) AS TotalBalance,
    COUNT(t.TransactionID) AS TransactionCount
FROM Branch b
LEFT JOIN Customer c ON b.BranchID = c.BranchID
LEFT JOIN Account a ON c.CustomerID = a.CustomerID
LEFT JOIN `Transaction` t ON a.AccountID = t.AccountID
GROUP BY b.BranchName;
```

3. Triggers (15 Marks) – Create trigger on transaction insert:

- TransactionAudit(TransactionID, AccountID, Amount, LogDate, Action)

```
CREATE TABLE TransactionAudit (
  AuditID INT PRIMARY KEY AUTO_INCREMENT,
  TransactionID INT,
  AccountID INT,
  Amount DECIMAL(15,2),
  LogDate DATETIME DEFAULT CURRENT_TIMESTAMP,
  Action VARCHAR(100)
);

DELIMITER $$
CREATE TRIGGER trg_after_transaction_insert
AFTER INSERT ON `Transaction`
FOR EACH ROW
BEGIN
  -- Log the transaction
  INSERT INTO TransactionAudit(TransactionID, AccountID, Amount, Action)
  VALUES (NEW.TransactionID, NEW.AccountID, NEW.Amount, CONCAT('Type: ', NEW.TransactionType));

  -- Update account balance for Deposit/Withdrawal
  IF NEW.TransactionType = 'Deposit' THEN
    UPDATE Account SET Balance = Balance + NEW.Amount WHERE AccountID = NEW.AccountID;
  ELSEIF NEW.TransactionType = 'Withdrawal' THEN
    UPDATE Account SET Balance = Balance - NEW.Amount WHERE AccountID = NEW.AccountID;
  END IF;
END$$
DELIMITER ;
```

4. Analytical Queries (15 Marks):

- Top 2 branches by total customer deposits

```
SELECT br.BranchName, ROUND(SUM(CASE WHEN t.TransactionType='Deposit' THEN t.Amount ELSE 0 END),2) AS TotalDeposits
FROM Branch br
JOIN Customer c ON br.BranchID = c.BranchID
JOIN Account a ON c.CustomerID = a.CustomerID
JOIN `Transaction` t ON a.AccountID = t.AccountID
GROUP BY br.BranchName
ORDER BY TotalDeposits DESC
LIMIT 2;
```

- Average account balance by account type

```
SELECT AccountType, ROUND(AVG(Balance),2) AS AvgBalance
```

```
FROM Account
GROUP BY AccountType;
```

- Customers with transaction count > 20 in current year

```
SELECT c.CustomerID, c.Name, COUNT(t.TransactionID) AS TxCount
FROM Customer c
JOIN Account a ON c.CustomerID = a.CustomerID
JOIN `Transaction` t ON a.AccountID = t.AccountID
WHERE YEAR(t.TransactionDate) = YEAR(CURDATE())
GROUP BY c.CustomerID, c.Name
HAVING COUNT(t.TransactionID) > 20;
```