**1. Library Management System**

**Table Structures**

CREATE TABLE Authors (

author\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

country VARCHAR(50)

);

CREATE TABLE Books (

book\_id INT PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(255) NOT NULL,

author\_id INT,

published\_year YEAR,

genre VARCHAR(50),

FOREIGN KEY (author\_id) REFERENCES Authors(author\_id) ON DELETE CASCADE

);

CREATE TABLE Members (

member\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

join\_date DATE DEFAULT CURDATE()

);

CREATE TABLE BorrowedBooks (

borrow\_id INT PRIMARY KEY AUTO\_INCREMENT,

book\_id INT,

member\_id INT,

borrow\_date DATE DEFAULT CURDATE(),

return\_date DATE,

FOREIGN KEY (book\_id) REFERENCES Books(book\_id) ON DELETE CASCADE,

FOREIGN KEY (member\_id) REFERENCES Members(member\_id) ON DELETE CASCADE

);

**Queries**

-- 1. Add new book

INSERT INTO Books (title, author\_id, published\_year, genre)

VALUES ('The Great Gatsby', 1, 1925, 'Fiction');

-- 2. List all borrowed books with members

SELECT b.title, m.name, bb.borrow\_date, bb.return\_date

FROM BorrowedBooks bb

JOIN Books b ON bb.book\_id = b.book\_id

JOIN Members m ON bb.member\_id = m.member\_id;

-- 3. Find overdue books

SELECT b.title, m.name, bb.borrow\_date, bb.return\_date

FROM BorrowedBooks bb

JOIN Books b ON bb.book\_id = b.book\_id

JOIN Members m ON bb.member\_id = m.member\_id

WHERE bb.return\_date < CURDATE() AND bb.return\_date IS NOT NULL;

**2. Employee Attendance System**

**Table Structures**

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

department VARCHAR(50),

email VARCHAR(100) UNIQUE NOT NULL

);

CREATE TABLE Attendance (

attendance\_id INT PRIMARY KEY AUTO\_INCREMENT,

emp\_id INT,

date DATE NOT NULL,

check\_in TIME NOT NULL,

check\_out TIME,

FOREIGN KEY (emp\_id) REFERENCES Employees(emp\_id) ON DELETE CASCADE

);

**Queries**

-- 1. Mark attendance

INSERT INTO Attendance (emp\_id, date, check\_in)

VALUES (1, CURDATE(), '09:00:00');

-- 2. Get attendance report for an employee

SELECT e.name, a.date, a.check\_in, a.check\_out

FROM Attendance a

JOIN Employees e ON a.emp\_id = e.emp\_id

WHERE e.emp\_id = 1;

-- 3. Find employees who are late (checked in after 9:30 AM)

SELECT e.name, a.date, a.check\_in

FROM Attendance a

JOIN Employees e ON a.emp\_id = e.emp\_id

WHERE a.check\_in > '09:30:00';

**3. Online Shopping System**

**Table Structures**

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(15)

);

CREATE TABLE Products (

product\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

price DECIMAL(10,2) NOT NULL,

stock INT NOT NULL

);

CREATE TABLE Orders (

order\_id INT PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT,

order\_date DATE DEFAULT CURDATE(),

total\_amount DECIMAL(10,2),

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id) ON DELETE CASCADE

);

CREATE TABLE OrderDetails (

order\_detail\_id INT PRIMARY KEY AUTO\_INCREMENT,

order\_id INT,

product\_id INT,

quantity INT NOT NULL,

FOREIGN KEY (order\_id) REFERENCES Orders(order\_id) ON DELETE CASCADE,

FOREIGN KEY (product\_id) REFERENCES Products(product\_id) ON DELETE CASCADE

);

**Queries**

-- 1. Place an order

INSERT INTO Orders (customer\_id, total\_amount)

VALUES (1, 100.50);

-- 2. Get order details for a customer

SELECT o.order\_id, o.order\_date, p.name, od.quantity, p.price

FROM Orders o

JOIN OrderDetails od ON o.order\_id = od.order\_id

JOIN Products p ON od.product\_id = p.product\_id

WHERE o.customer\_id = 1;

-- 3. List best-selling products

SELECT p.name, SUM(od.quantity) AS total\_sold

FROM OrderDetails od

JOIN Products p ON od.product\_id = p.product\_id

GROUP BY p.name

ORDER BY total\_sold DESC

LIMIT 5;

**4. Hospital Management System**

**Table Structures**

CREATE TABLE Doctors (

doctor\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

specialty VARCHAR(50)

);

CREATE TABLE Patients (

patient\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

age INT NOT NULL,

gender ENUM('Male', 'Female', 'Other'),

contact VARCHAR(15)

);

CREATE TABLE Appointments (

appointment\_id INT PRIMARY KEY AUTO\_INCREMENT,

patient\_id INT,

doctor\_id INT,

appointment\_date DATETIME NOT NULL,

status ENUM('Scheduled', 'Completed', 'Cancelled') DEFAULT 'Scheduled',

FOREIGN KEY (patient\_id) REFERENCES Patients(patient\_id) ON DELETE CASCADE,

FOREIGN KEY (doctor\_id) REFERENCES Doctors(doctor\_id) ON DELETE CASCADE

);

**Queries**

-- 1. Schedule an appointment

INSERT INTO Appointments (patient\_id, doctor\_id, appointment\_date)

VALUES (1, 2, '2025-02-10 10:30:00');

-- 2. List all appointments with doctors

SELECT a.appointment\_id, p.name AS patient, d.name AS doctor, a.appointment\_date, a.status

FROM Appointments a

JOIN Patients p ON a.patient\_id = p.patient\_id

JOIN Doctors d ON a.doctor\_id = d.doctor\_id;

-- 3. Find all upcoming appointments for a doctor

SELECT a.appointment\_id, p.name AS patient, a.appointment\_date

FROM Appointments a

JOIN Patients p ON a.patient\_id = p.patient\_id

WHERE a.doctor\_id = 2 AND a.appointment\_date > NOW();

**5. Student Management System**

**Table Structures**

CREATE TABLE Students (

student\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

date\_of\_birth DATE NOT NULL

);

CREATE TABLE Courses (

course\_id INT PRIMARY KEY AUTO\_INCREMENT,

course\_name VARCHAR(100) NOT NULL,

credits INT NOT NULL

);

CREATE TABLE Enrollments (

enrollment\_id INT PRIMARY KEY AUTO\_INCREMENT,

student\_id INT,

course\_id INT,

enrollment\_date DATE DEFAULT CURDATE(),

grade CHAR(2),

FOREIGN KEY (student\_id) REFERENCES Students(student\_id) ON DELETE CASCADE,

FOREIGN KEY (course\_id) REFERENCES Courses(course\_id) ON DELETE CASCADE

);

**Queries**

-- 1. Enroll a student in a course

INSERT INTO Enrollments (student\_id, course\_id, grade)

VALUES (1, 3, 'A');

-- 2. Get all students enrolled in a course

SELECT s.name, c.course\_name, e.grade

FROM Enrollments e

JOIN Students s ON e.student\_id = s.student\_id

JOIN Courses c ON e.course\_id = c.course\_id

WHERE e.course\_id = 3;

-- 3. Find students with highest grades in each course

SELECT c.course\_name, s.name, e.grade

FROM Enrollments e

JOIN Students s ON e.student\_id = s.student\_id

JOIN Courses c ON e.course\_id = c.course\_id

WHERE e.grade = 'A';