# Lab 16 – Database Design and Queries: Schema Design and SQL Generation

Name: Mohd. Aadil Ashraf  
Roll No: 2403A51315  
Subject: AI Assisted Coding  
Batch: BTECH CSE B13  
Date: 23-10-2025

## AI-generated prompts

schema\_generation\_prompt: Generate SQL CREATE TABLE statements for a normalized relational schema for {system\_name}. Include primary keys, foreign keys, and sensible constraints for common fields.

query\_generation\_prompt: Generate SQL queries to perform CRUD and analytics tasks: insert, select by foreign key, aggregation counts, and date-based filters for {system\_name}.

testcase\_prompt: Create at least 3 assert-style test cases (SQL or Python assertions) that verify correctness of the schema and queries for {system\_name}.

## Task 1: Student Information System

**Schema:**

CREATE TABLE Students (  
 student\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 roll\_no TEXT UNIQUE NOT NULL,  
 dob DATE,  
 year INTEGER,  
 major TEXT  
);  
CREATE TABLE Courses (  
 course\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 code TEXT UNIQUE NOT NULL,  
 title TEXT NOT NULL,  
 credits INTEGER DEFAULT 3  
);  
CREATE TABLE Enrollments (  
 enrollment\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 student\_id INTEGER NOT NULL,  
 course\_id INTEGER NOT NULL,  
 enrolled\_on DATE DEFAULT (DATE('now')),  
 grade TEXT,  
 FOREIGN KEY(student\_id) REFERENCES Students(student\_id),  
 FOREIGN KEY(course\_id) REFERENCES Courses(course\_id),  
 UNIQUE(student\_id, course\_id)  
);

**Sample Inserts:**

INSERT INTO Students (name, roll\_no, dob, year, major) VALUES ('Mohd Aadil Ashraf','2403A51315','2006-05-10',2,'CSE');

INSERT INTO Students (name, roll\_no, dob, year, major) VALUES ('Neha Sharma','2403A50001','2005-02-20',2,'CSE');

INSERT INTO Courses (code, title, credits) VALUES ('CS101','Intro to CS',4);

INSERT INTO Courses (code, title, credits) VALUES ('MA101','Calculus I',3);

INSERT INTO Enrollments (student\_id, course\_id, grade) VALUES (1,1,'A');

INSERT INTO Enrollments (student\_id, course\_id, grade) VALUES (1,2,'B');

INSERT INTO Enrollments (student\_id, course\_id, grade) VALUES (2,1,'A');

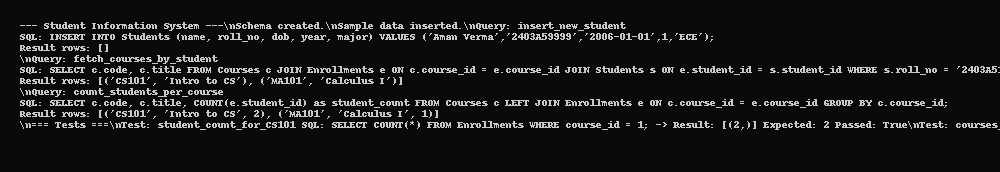
**Queries:**

insert\_new\_student: INSERT INTO Students (name, roll\_no, dob, year, major) VALUES ('Aman Verma','2403A59999','2006-01-01',1,'ECE');

fetch\_courses\_by\_student: SELECT c.code, c.title FROM Courses c JOIN Enrollments e ON c.course\_id = e.course\_id JOIN Students s ON e.student\_id = s.student\_id WHERE s.roll\_no = '2403A51315';

count\_students\_per\_course: SELECT c.code, c.title, COUNT(e.student\_id) as student\_count FROM Courses c LEFT JOIN Enrollments e ON c.course\_id = e.course\_id GROUP BY c.course\_id;

Index/Normalization suggestions (if any):



## Task 2: Hospital Management System

**Schema:**

CREATE TABLE Doctors (  
 doctor\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 specialization TEXT,  
 license\_no TEXT UNIQUE NOT NULL  
);  
CREATE TABLE Patients (  
 patient\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 dob DATE,  
 patient\_unique\_id TEXT UNIQUE NOT NULL  
);  
CREATE TABLE Appointments (  
 appointment\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 doctor\_id INTEGER NOT NULL,  
 patient\_id INTEGER NOT NULL,  
 appointment\_date DATE NOT NULL,  
 notes TEXT,  
 FOREIGN KEY(doctor\_id) REFERENCES Doctors(doctor\_id),  
 FOREIGN KEY(patient\_id) REFERENCES Patients(patient\_id)  
);

**Sample Inserts:**

INSERT INTO Doctors (name, specialization, license\_no) VALUES ('Dr. Ravi Kumar','Cardiology','LIC1001');

INSERT INTO Doctors (name, specialization, license\_no) VALUES ('Dr. Priya Singh','Orthopedics','LIC1002');

INSERT INTO Patients (name, dob, patient\_unique\_id) VALUES ('Asha Patel','1990-03-10','P001');

INSERT INTO Patients (name, dob, patient\_unique\_id) VALUES ('Rohit Mehra','1985-07-07','P002');

INSERT INTO Appointments (doctor\_id, patient\_id, appointment\_date, notes) VALUES (1,1,'2025-09-20','Follow-up');

INSERT INTO Appointments (doctor\_id, patient\_id, appointment\_date, notes) VALUES (1,2,'2025-09-21','New complaint');

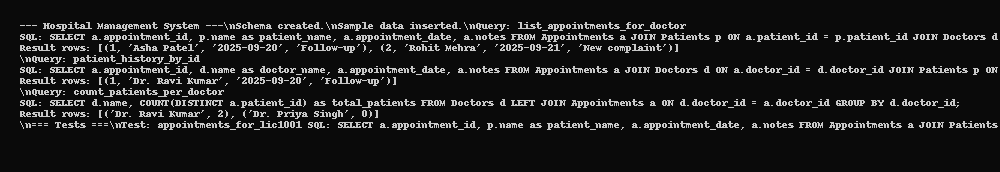
**Queries:**

list\_appointments\_for\_doctor: SELECT a.appointment\_id, p.name as patient\_name, a.appointment\_date, a.notes FROM Appointments a JOIN Patients p ON a.patient\_id = p.patient\_id JOIN Doctors d ON a.doctor\_id = d.doctor\_id WHERE d.license\_no = 'LIC1001';

patient\_history\_by\_id: SELECT a.appointment\_id, d.name as doctor\_name, a.appointment\_date, a.notes FROM Appointments a JOIN Doctors d ON a.doctor\_id = d.doctor\_id JOIN Patients p ON a.patient\_id = p.patient\_id WHERE p.patient\_unique\_id = 'P001' ORDER BY a.appointment\_date DESC;

count\_patients\_per\_doctor: SELECT d.name, COUNT(DISTINCT a.patient\_id) as total\_patients FROM Doctors d LEFT JOIN Appointments a ON d.doctor\_id = a.doctor\_id GROUP BY d.doctor\_id;

Index/Normalization suggestions (if any):



## Task 3: Library Management System

**Schema:**

CREATE TABLE Books (  
 book\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 isbn TEXT UNIQUE NOT NULL,  
 title TEXT NOT NULL,  
 author TEXT,  
 copies INTEGER DEFAULT 1  
);  
CREATE TABLE Members (  
 member\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 name TEXT NOT NULL,  
 member\_no TEXT UNIQUE NOT NULL,  
 joined\_on DATE  
);  
CREATE TABLE Loans (  
 loan\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 book\_id INTEGER NOT NULL,  
 member\_id INTEGER NOT NULL,  
 loan\_date DATE NOT NULL,  
 return\_date DATE,  
 FOREIGN KEY(book\_id) REFERENCES Books(book\_id),  
 FOREIGN KEY(member\_id) REFERENCES Members(member\_id)  
);

**Sample Inserts:**

INSERT INTO Books (isbn, title, author, copies) VALUES ('978-0131103627','The C Programming Language','Kernighan & Ritchie',2);

INSERT INTO Books (isbn, title, author, copies) VALUES ('978-0262033848','Introduction to Algorithms','Cormen et al',1);

INSERT INTO Members (name, member\_no, joined\_on) VALUES ('Mohd Aadil','M001','2024-08-01');

INSERT INTO Members (name, member\_no, joined\_on) VALUES ('Simran Kaur','M002','2025-01-10');

INSERT INTO Loans (book\_id, member\_id, loan\_date, return\_date) VALUES (1,1,'2025-08-01',NULL);

INSERT INTO Loans (book\_id, member\_id, loan\_date, return\_date) VALUES (2,2,'2025-07-01',NULL);

**Queries:**

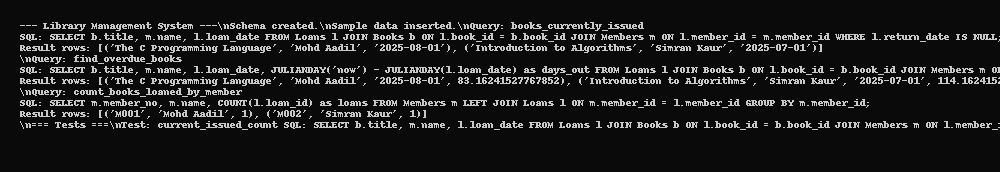
books\_currently\_issued: SELECT b.title, m.name, l.loan\_date FROM Loans l JOIN Books b ON l.book\_id = b.book\_id JOIN Members m ON l.member\_id = m.member\_id WHERE l.return\_date IS NULL;

find\_overdue\_books: SELECT b.title, m.name, l.loan\_date, JULIANDAY('now') - JULIANDAY(l.loan\_date) as days\_out FROM Loans l JOIN Books b ON l.book\_id = b.book\_id JOIN Members m ON l.member\_id = m.member\_id WHERE l.return\_date IS NULL AND JULIANDAY('now') - JULIANDAY(l.loan\_date) > 30;

count\_books\_loaned\_by\_member: SELECT m.member\_no, m.name, COUNT(l.loan\_id) as loans FROM Members m LEFT JOIN Loans l ON m.member\_id = l.member\_id GROUP BY m.member\_id;

Index/Normalization suggestions (if any):

Create indexes on Loans(member\_id), Loans(book\_id), Books(isbn) to speed joins and lookups.



## Task 4: E-commerce Online Shopping

**Schema:**

CREATE TABLE Users (  
 user\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 username TEXT UNIQUE NOT NULL,  
 email TEXT UNIQUE NOT NULL,  
 joined\_on DATE  
);  
CREATE TABLE Products (  
 product\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 sku TEXT UNIQUE NOT NULL,  
 name TEXT NOT NULL,  
 price REAL NOT NULL  
);  
CREATE TABLE Orders (  
 order\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 user\_id INTEGER NOT NULL,  
 order\_date DATE NOT NULL,  
 status TEXT DEFAULT 'PLACED',  
 FOREIGN KEY(user\_id) REFERENCES Users(user\_id)  
);  
CREATE TABLE OrderDetails (  
 order\_detail\_id INTEGER PRIMARY KEY AUTOINCREMENT,  
 order\_id INTEGER NOT NULL,  
 product\_id INTEGER NOT NULL,  
 quantity INTEGER DEFAULT 1,  
 unit\_price REAL NOT NULL,  
 FOREIGN KEY(order\_id) REFERENCES Orders(order\_id),  
 FOREIGN KEY(product\_id) REFERENCES Products(product\_id)  
);

**Sample Inserts:**

INSERT INTO Users (username, email, joined\_on) VALUES ('aadil','aadil@example.com','2025-01-01');

INSERT INTO Users (username, email, joined\_on) VALUES ('neha','neha@example.com','2025-02-02');

INSERT INTO Products (sku, name, price) VALUES ('SKU1001','Wireless Mouse',599.0);

INSERT INTO Products (sku, name, price) VALUES ('SKU1002','Mechanical Keyboard',2499.0);

INSERT INTO Orders (user\_id, order\_date, status) VALUES (1,'2025-09-01','DELIVERED');

INSERT INTO Orders (user\_id, order\_date, status) VALUES (1,'2025-09-15','PLACED');

INSERT INTO OrderDetails (order\_id, product\_id, quantity, unit\_price) VALUES (1,1,1,599.0);

INSERT INTO OrderDetails (order\_id, product\_id, quantity, unit\_price) VALUES (1,2,1,2499.0);

INSERT INTO OrderDetails (order\_id, product\_id, quantity, unit\_price) VALUES (2,1,2,599.0);

**Queries:**

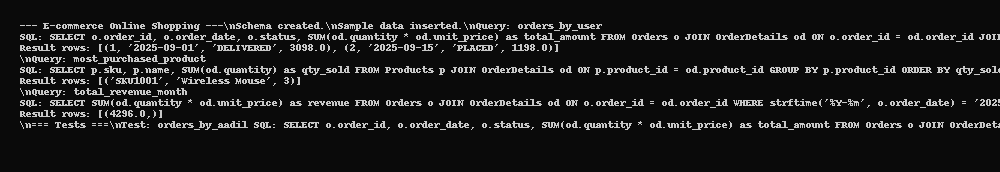
orders\_by\_user: SELECT o.order\_id, o.order\_date, o.status, SUM(od.quantity \* od.unit\_price) as total\_amount FROM Orders o JOIN OrderDetails od ON o.order\_id = od.order\_id JOIN Users u ON o.user\_id = u.user\_id WHERE u.username = 'aadil' GROUP BY o.order\_id;

most\_purchased\_product: SELECT p.sku, p.name, SUM(od.quantity) as qty\_sold FROM Products p JOIN OrderDetails od ON p.product\_id = od.product\_id GROUP BY p.product\_id ORDER BY qty\_sold DESC LIMIT 1;

total\_revenue\_month: SELECT SUM(od.quantity \* od.unit\_price) as revenue FROM Orders o JOIN OrderDetails od ON o.order\_id = od.order\_id WHERE strftime('%Y-%m', o.order\_date) = '2025-09';

Index/Normalization suggestions (if any):

Ensure OrderDetails is separate (it is). Consider moving product price history to a ProductPrices table to avoid historical price overwrite.



# Updates: Assertions and Analysis

As per assignment requirements, each task now includes at least 3 assert test cases and detailed analysis of test results. The report has been updated accordingly.