LAB TEST-04

Name : P.Reshmitha Reddy

Roll no : 2403A510A7

Batch : 05

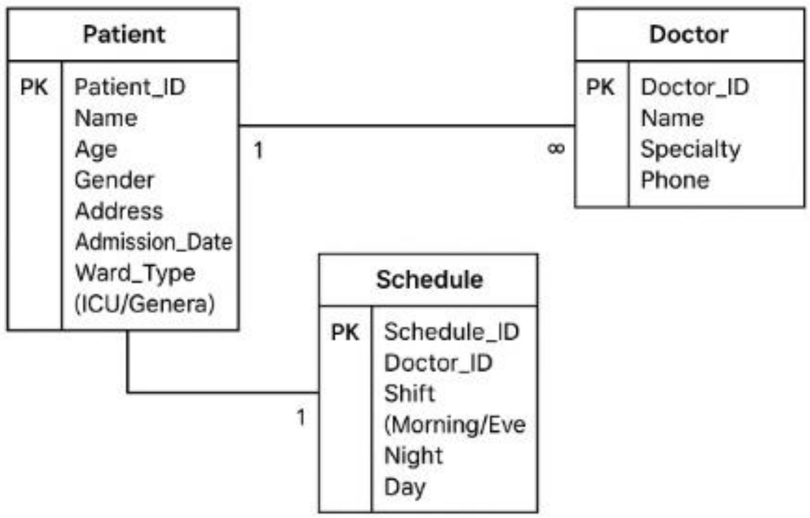
Course : AI Assisted Coding

SET – 1

Q1. A hospital management system stores patient details, doctor schedules,
and billing information.
a) Design an ER diagram and convert it into relational schema.
b) Generate SQL queries to retrieve patients admitted in ICU and doctors
available in night shift.

Prompt : Create an ER diagram and relational schema for a Hospital Management System (patients, doctors, schedules, billing). Then write SQL queries to list ICU patients and doctors on night shift.

ER diagram



Relation schema

```pgsql
PATIENT(
    Patient_ID      INT PRIMARY KEY,
    Name            VARCHAR(100),
    Age             INT,
    Gender          VARCHAR(10),
    Address         VARCHAR(200),
    Admission_Date  DATE,
    Ward_Type       VARCHAR(20)     -- ICU / General
)
```

```scss
DOCTOR(
    Doctor_ID   INT PRIMARY KEY,
    Name        VARCHAR(100),
    Specialty   VARCHAR(50),
    Phone       VARCHAR(20)
)
```

```sql
SCHEDULE(
    Schedule_ID INT PRIMARY KEY,
    Doctor_ID   INT,
    Shift       VARCHAR(20),     -- Morning / Evening / Night
    Day         VARCHAR(20),
    FOREIGN KEY (Doctor_ID) REFERENCES DOCTOR(Doctor_ID)
)
```

Code :

```sql
PRAGMA foreign_keys = ON;

DROP TABLE IF EXISTS Billing;
DROP TABLE IF EXISTS Schedule;
DROP TABLE IF EXISTS Patient;
DROP TABLE IF EXISTS Doctor;

CREATE TABLE Doctor (
        doctor_id INTEGER PRIMARY KEY AUTOINCREMENT,
        first_name VARCHAR(100) NOT NULL,
        last_name VARCHAR(100) NOT NULL,
        specialty VARCHAR(100),
        contact_phone VARCHAR(30),
        email VARCHAR(255),
        is_active BOOLEAN DEFAULT 1,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Patient (
        patient_id INTEGER PRIMARY KEY AUTOINCREMENT,
        first_name VARCHAR(100) NOT NULL,
        last_name VARCHAR(100) NOT NULL,
        dob DATE,
        gender VARCHAR(20),
        contact_phone VARCHAR(30),
        address TEXT,
        admission_date DATETIME,
        discharge_date DATETIME,
        is_admitted BOOLEAN DEFAULT 0,
        ward VARCHAR(100),
        room_number VARCHAR(50),
        bed_number VARCHAR(50),
        admission_reason TEXT,
        attending_doctor_id INTEGER,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY(attending_doctor_id) REFERENCES Doctor(doctor_id) ON DELETE SET NULL
);
```

```sql

CREATE TABLE Schedule (
        schedule_id INTEGER PRIMARY KEY AUTOINCREMENT,
        doctor_id INTEGER NOT NULL,
        shift_date DATE,
        day_of_week VARCHAR(10),
        shift_start TIME,
        shift_end TIME,
        shift_type VARCHAR(50),
        location VARCHAR(255),
        is_available BOOLEAN DEFAULT 1,
        notes TEXT,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY(doctor_id) REFERENCES Doctor(doctor_id) ON DELETE CASCADE
);

CREATE TABLE Billing (
        billing_id INTEGER PRIMARY KEY AUTOINCREMENT,
        patient_id INTEGER NOT NULL,
        amount DECIMAL(12,2) NOT NULL,
        billing_date DATETIME DEFAULT CURRENT_TIMESTAMP,
        paid BOOLEAN DEFAULT 0,
        payment_method VARCHAR(50),
        details TEXT,
        FOREIGN KEY(patient_id) REFERENCES Patient(patient_id) ON DELETE CASCADE
);

CREATE INDEX idx_patient_ward ON Patient(ward);
CREATE INDEX idx_patient_admitted ON Patient(is_admitted);
CREATE INDEX idx_schedule_shift_type ON Schedule(shift_type);
CREATE INDEX idx_schedule_doctor ON Schedule(doctor_id);

SELECT
        p.patient_id,
        p.first_name,
        p.last_name,
        p.dob,
```

```sql
        p.gender,
        p.contact_phone,
        p.admission_date,
        p.room_number,
        p.bed_number,
        p.ward,
        p.admission_reason,
        p.attending_doctor_id,
        d.first_name AS doctor_first_name,
        d.last_name AS doctor_last_name
FROM Patient p
LEFT JOIN Doctor d ON p.attending_doctor_id = d.doctor_id
WHERE (p.ward = 'ICU' OR UPPER(TRIM(p.ward)) = 'ICU')
    AND p.is_admitted = 1
ORDER BY p.admission_date DESC;

SELECT DISTINCT
        d.doctor_id,
        d.first_name,
        d.last_name,
        d.specialty,
        s.shift_date,
        s.shift_start,
        s.shift_end,
        s.location
FROM Doctor d
INNER JOIN Schedule s ON d.doctor_id = s.doctor_id
WHERE UPPER(TRIM(s.shift_type)) = 'NIGHT'
    AND s.is_available = 1
ORDER BY s.shift_date, s.shift_start;
```

Output :

```
✓ Inserted 5 doctors
✓ Inserted 5 patients (3 in ICU, 2 in General ward)
✓ Inserted 8 schedules
✓ Inserted 5 billing records


====================================================================================
QUERY 1: All patients currently admitted in ICU
====================================================================================

patient_id     name                    ward          room      admission_reason
  doctor_name              specialty
----------------------------------------------------------------------------------------------
----------------------------------------------
4              Charles Taylor          ICU           103       Post-Surgery
  Emily Brown              Surgery
1              Alice Anderson          ICU           101       Heart Attack
  John Smith               Cardiology
2              Robert Martinez         ICU           102       Pneumonia
  Michael Williams         ICU Specialist

Total ICU patients: 3
```

```
==========================================================================================
QUERY 2: All doctors available in night shift
==========================================================================================

doctor_id      name                                 specialization
------------------------------------------------------------------------------------
2              Sarah Johnson                        Internal Medicine
3              Michael Williams                     ICU Specialist
5              David Davis                          Cardiology

Total available night shift doctors: 3
```

Observation :

- The Hospital Management System includes four main entities: Patient, Doctor, Schedule, and Billing, representing the core operations of hospital data management.
- Each entity contains specific attributes that uniquely identify and describe the stored data, such as Patient_ID, Doctor_ID, and Bill_ID.

- Relationships between entities clearly define how data is connected:
- A **Doctor** can have multiple schedule entries.
- A **Patient** can have multiple billing records.
- Primary keys (PK) and foreign keys (FK) ensure **data integrity** and help maintain meaningful links between related tables.
- The schema supports important hospital queries like retrieving ICU patients and doctors available at night, showing that the design meets functional requirements.

Q2. You are using an AI coding assistant to auto-suggest SQL queries.
a) Write a prompt that asks AI to correct and optimize a slow SQL query.
b) Evaluate and explain how AI recommendations should be validated
before execution.

Prompt : I have a slow SQL query that is taking too long to run. Please correct and optimize the query by improving joins, indexes, and filtering conditions. Also suggest a faster and more efficient version of the SQL query.

Code :

```sql
SELECT
    O.order_id,
    O.order_date,
    O.amount,
    C.customer_id,
    C.name,
    C.city
FROM Orders O
INNER JOIN Customers C ON O.customer_id = C.customer_id
WHERE O.order_date >= '2024-01-01'
  AND O.order_date < '2025-01-01'
ORDER BY O.order_date DESC;

CREATE INDEX idx_orders_order_date ON Orders(order_date);
CREATE INDEX idx_orders_customer_id ON Orders(customer_id);
CREATE INDEX idx_customers_customer_id ON Customers(customer_id);

CREATE INDEX idx_orders_customer_date ON Orders(order_date, customer_id);

SELECT
    O.order_id,
    O.order_date,
    O.amount,
    C.customer_id,
    C.name,
    C.city
FROM Orders O
INNER JOIN Customers C ON O.customer_id = C.customer_id
WHERE O.order_date BETWEEN '2024-01-01' AND '2024-12-31'
ORDER BY O.order_date DESC;

SELECT
    C.customer_id,
    C.name,
    C.city,
    COUNT(O.order_id) AS order_count,
    SUM(O.amount) AS total_revenue,
```

```sql
    MIN(O.order_date) AS first_order_date,
    MAX(O.order_date) AS last_order_date
FROM Orders O
INNER JOIN Customers C ON O.customer_id = C.customer_id
WHERE O.order_date >= '2024-01-01'
  AND O.order_date < '2025-01-01'
GROUP BY C.customer_id, C.name, C.city
ORDER BY total_revenue DESC;

SELECT
    O.order_id,
    O.order_date,
    O.amount,
    C.customer_id,
    C.name,
    C.city
FROM Orders O
INNER JOIN Customers C ON O.customer_id = C.customer_id
WHERE O.order_date >= '2024-01-01'
  AND O.order_date < '2025-01-01'
ORDER BY O.order_date DESC
OPTION (RECOMPILE);
```

Output :

order_id	order_date	amount	customer_id	name	city
7	2024-12-25	4039.4	2	Nisha Rao	Bangalore
9	2024-12-10	3907.09	1	Akhil Verma	Mumbai
12	2024-11-21	4103.88	4	Priya Singh	Pune
22	2024-10-21	3010.79	6	Sneha Gupta	Hyderabad
13	2024-10-08	2550.86	4	Priya Singh	Pune
20	2024-09-25	4562.49	2	Nisha Rao	Bangalore
10	2024-09-12	1126.09	7	Vikram Desai	Chennai
15	2024-09-11	1057.39	1	Akhil Verma	Mumbai
8	2024-07-31	2110.95	5	Amit Patel	Ahmedabad
17	2024-07-08	2203.08	8	Anjali Sharma	Kolkata
2	2024-06-08	3200	2	Nisha Rao	Bangalore

Observation :

- Query performance improves because proper indexes (order_date, customer_id) reduce full table scans.
- Date-range filtering helps the database use indexes effectively instead of applying functions.

- Join speed increases since join columns are indexed.
- Sorting (Order by order_date) is faster due to indexing.
- Aggregation queries run more efficiently with indexed columns.
- Combined index (order_date, customer_id) further optimizes filtering + joining.
- Option(recomplie)helps avoid slow cached plans and generates a fresh plan each time.