Project 4: Medical Laboratory System Database Design and Implementation

Sarah Yaser 221001902

Mohamed Reda 221001846

Mohamed KHalaf 222000072

Asmaa Ragab 221001773

**Dr. Mohamed ELSayeh**

**1. Introduction**

This project is about making a database system for a medical laboratory. It keeps track of laboratorians, patients, medical tests, test results, and test parts. The system helps the lab work better by managing test materials, recording results correctly, and making it easy to find patient test information. Also, a simple desktop app was made using Python and Tkinter so users can manage the data easily without using complicated commands

**2. Entities and Attributes**

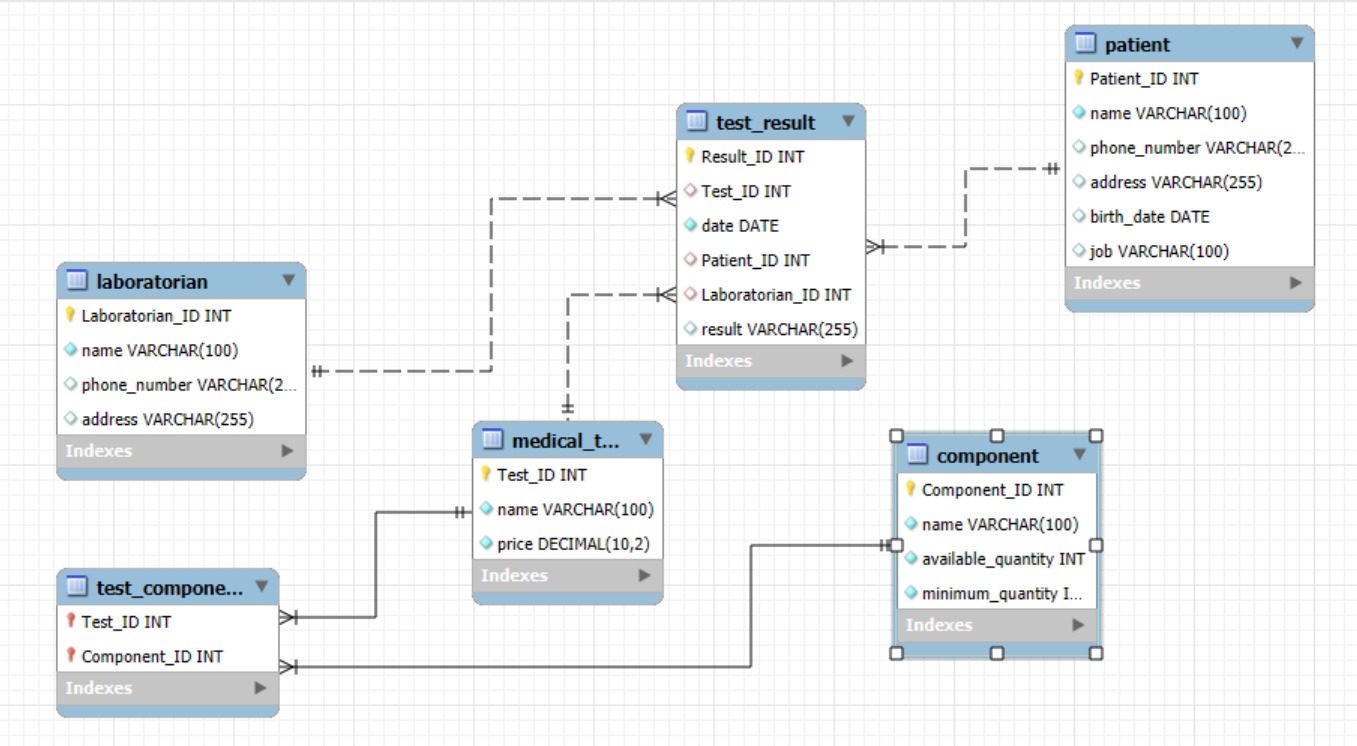
**The system contains the following main entities and their attributes:**

* Laboratorian: ID (Primary Key), Name, Phone Number, Address
* Patient: ID (Primary Key), Name, Phone Number, Address, Birth Date, Job

Component: ID (Primary Key), Name, Available Quantity, Minimum Quantity

* Medical Test: Test ID (Primary Key), Name, Price
* Test Component: Associative entity linking Medical Test and Component, with Test ID and Component ID as foreign keys (Composite Primary Key)
* Test Result: Result ID (Primary Key), Test ID (Foreign Key), Date, Patient ID (Foreign Key), Laboratorian ID (Foreign Key), Result Description

**3. ER Diagram Explanation**



The ER diagram shows the main relationships in the system.

* A one-to-many relationship between Laboratorian and Test Result (a laboratorian can perform many test results).
* A one-to-many relationship between Patient and Test Result (a patient can have multiple test results).
* A one-to-many relationship between medical Test and Test Result
* A one-to-many relationship between Medical Test and Test Component
* A one-to-many relationship between Component and Test Component
* A many-to-many relationship between Medical Test and Component, managed through the Test Component associative entity.
* Test Result requires total participation from Patient and Laboratorian. Components have partial participation with Medical Test.

**4. Relational Schema**

The relational schema for the Medical Laboratory System includes the following tables with their attributes and constraints

|  |  |  |
| --- | --- | --- |
| **Table** | **Attributes** | **Keys** |
| **Laboratorian** | Laboratorian\_ID, Name, Phone\_Number, Address | PK: Laboratorian\_ID |
| **Patient** | Patient\_ID, Name, Phone\_Number, Address, Birth\_Date, Job | PK: Patient\_ID |
| **Component** | Component\_ID, Name, Available Quantity, Minimum\_Quantity | PK: Component\_ID |
| **Medical\_Test** | Test\_ID, Name, Price | PK: Test\_ID |
| **Test\_Component** | Test\_ID, Component\_ID | PK: (Test\_ID, Component\_ID) FK: to Medical\_Test and Component |
| **Test\_Result** | Result\_ID, Test\_ID, Date, Patient\_ID, Laboratorian\_ID, Result | PK: Result\_ID FK: to related tables |

**Test\_Result Table**

* Result\_ID (INT) — Primary Key
* Test\_ID (INT) — Foreign Key: references Medical\_Test(Test\_ID)
* Patient\_ID (INT) — Foreign Key: references Patient (Patient\_ID)
* Laboratorian\_ID (INT) — Foreign Key: references Laboratorian (Laboratorian\_ID)
* Date (DATE)

**Explanation:**

The schema organizes the data into tables with clear relationships. Each table has a **primary key** to identify records, and **foreign keys** to link related tables. The Test\_Component table uses a **composite key** to handle the many-to-many relationship between tests and components.

This design helps store, update, and retrieve lab data accurately while keeping relationships clear and consistent

**5. SQL Statements for Table Creation**

CREATE TABLE Laboratorian (

Laboratorian\_ID INT PRIMARY KEY,

name VARCHAR (100) NOT NULL,

Phone-number VARCHAR (20),

address VARCHAR (255)

);

CREATE TABLE Patient (

Patient-ID INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

phone-number VARCHAR(20),

address VARCHAR(255),

birth-date DATE,

job VARCHAR (100)

);

CREATE TABLE Component (

Component-ID INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

available-quantity INT NOT NULL,

minimum\_quantity INT NOT NULL

);

CREATE TABLE Medical\_Test (

Test\_ID INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

price DECIMAL(10,2) NOT NULL

);

CREATE TABLE Test\_Component (

Test\_ID INT,

Component\_ID INT,

PRIMARY KEY (Test\_ID, Component\_ID),

FOREIGN KEY (Test\_ID) REFERENCES Medical\_Test(Test\_ID),

FOREIGN KEY (Component\_ID) REFERENCES Component(Component\_ID)

);

CREATE TABLE Test\_Result (

Result\_ID INT PRIMARY KEY,

Test\_ID INT,

date DATE NOT NULL,

Patient\_ID INT,

Laboratorian\_ID INT,

result VARCHAR (255), FOREIGN KEY (Test\_ID) REFERENCES Medical\_Test(Test\_ID), FOREIGN KEY (Patient\_ID) REFERENCES Patient (Patient\_ID),

FOREIGN KEY (Laboratorian\_ID) REFERENCES Laboratorian (Laboratorian\_ID)

);

**Explanation:**

The SQL statements create the tables for the medical lab system. Each table has a primary key to keep records unique and organized.

In Test\_Component, I used a composite key (Test\_ID and Component\_ID) to show the many-to-many relationship between tests and components.

I also used foreign keys to link related tables and keep the data consistent. Important fields include NOT NULL to make sure no required data is left out.

**6.Sample Data Insertion**

-- Insert sample laboratorians

INSERT INTO Laboratorian VALUES

(1, 'Ahmed Ali', '0123456789', 'Cairo'),

(2, 'Mona Hassan', '0112233445', 'Alexandria'),

(3, 'Omar Saeed', '0156789456', 'Giza'),

(4, 'Sara Khaled', '0109988776', 'Cairo'),

(5, 'Ali Mahmoud', '0123344556', 'Tanta'),

(6, 'Hana Youssef', '0115566778', 'Mansoura'),

(7, 'Khaled Farag', '0101234567', 'Cairo'),

(8, 'Nadia Ibrahim', '0129876543', 'Alexandria'),

(9, 'Tamer Adel', '0111112222', 'Cairo'),

(10, 'Dina Samir', '0122223333', 'Giza');

-- Insert sample patients

INSERT INTO Patient VALUES

(1001, 'Mohamed Salah', '0111111111', 'Cairo', '1985-07-15', 'Engineer'),

(1002, 'Fatma Ahmed', '0122222222', 'Alexandria', '1990-03-20', 'Teacher'),

(1003, 'Hany Mahmoud', '0103333333', 'Giza', '1975-11-05', 'Doctor'),

(1004, 'Laila Hassan', '0114444444', 'Cairo', '1988-06-30', 'Nurse'),

(1005, 'Ali Mohamed', '0125555555', 'Tanta', '1992-01-10', 'Student'),

(1006, 'Sara Khaled', '0106666666', 'Mansoura', '1980-12-25', 'Lawyer'),

(1007, 'Ahmed Youssef', '0117777777', 'Cairo', '1987-09-15', 'Accountant'),

(1008, 'Nora Sami', '0128888888', 'Alexandria', '1995-05-22', 'Designer'),

(1009, 'Tarek Adel', '0109999999', 'Giza', '1978-08-18', 'Architect'),

(1010, 'Dina Samir', '0110000000', 'Cairo', '1993-04-02', 'Pharmacist');

-

- Insert sample components

INSERT INTO Component VALUES

(1, 'Blood Sample Tube', 50, 10),

(2, 'Glucose Reagent', 30, 15),

(3, 'Hemoglobin Reagent', 40, 20),

(4, 'Microscope Slides', 60, 25),

(5, 'Urine Sample Cup', 45, 15),

(6, 'Chemical Reagent A', 20, 10),

(7, 'Chemical Reagent B', 15, 10),

(8, 'Test Tubes', 35, 10),

(9, 'Alcohol Swabs', 80, 30),

(10, 'Gloves', 100, 50);

-- Insert sample medical tests

INSERT INTO Medical\_Test VALUES

(101, 'CBC', 150.00),

(102, 'Blood Sugar', 120.00),

(103, 'Urinalysis', 130.00),

(104, 'Liver Function Test', 200.00)

, (105, 'Kidney Function Test', 190.00),

(106, 'Lipid Profile', 180.00),

(107, 'Thyroid Test', 170.00),

(108, 'Vitamin D Test', 160.00),

(109, 'COVID-19 PCR', 300.00),

(110, 'Electrolyte Panel', 140.00);

-- Insert sample test components

INSERT INTO Test\_Component VALUES

(101, 1),

(101, 3),

(102, 2),

(103, 5),

(104, 6),

(105, 7),

(106, 8),

(107, 9),

(108, 10),

(109, 4),

(110, 3);

-- Insert sample test results

INSERT INTO Test\_Result VALUES

(1, 101, '2024-04-01', 1001, 1, 'Normal CBC results'),

(2, 102, '2024-04-03', 1002, 2, 'High blood sugar level'),

(3, 103, '2024-03-30', 1003, '3', 'Normal urine analysis'),

(4, 104, '2024-04-05', 1004, 4, 'Elevated liver enzymes'),

(5, 105, '2024-04-07', 1005, 5, 'Normal kidney function'),

(6, 106, '2024-04-09', 1006, 6, 'High cholesterol'),

(7, 107, '2024-04-11', 1007, 7, 'Normal thyroid levels'),

(8, 108, '2024-04-13', 1008, 8, 'Vitamin D deficiency'),

(9, 109, '2024-04-15', 1009, 9, 'Negative COVID-19 PCR'),

(10, 110, '2024-04-17', 1010, 10, 'Normal electrolyte levels');

**Explanation:**

These SQL insertion commands add realistic sample data to each table, allowing the database to be tested effectively.

Having at least ten records per table ensures enough data variety to test queries and system functionality

**7.Example Queries**

List patients who performed the CBC test in the last year

List components whose available quantity is less than the minimum quantity

Calculate the total amount paid by a specific patient in the last 3 years

-- 1. List the names of patients who performed the CBC test in the last year SELECT DISTINCT p.name

FROM Patient p

JOIN Test\_Result tr ON p.Patient\_ID = tr.Patient\_ID

JOIN Medical\_Test mt ON tr.Test\_ID = mt.Test\_ID

WHERE mt.name = 'CBC' AND tr.date >= DATE\_SUB(CURDATE(), INTERVAL 1 YEAR);

-- 2. List the names of components where the available quantity is less than the minimum quantity

SELECT name, available\_quantity, minimum\_quantity

FROM Component

WHERE available\_quantity < minimum\_quantity;

-- 3. Calculate the total amount paid by a patient with Patient\_ID = 1001 in the last 3 years   
SELECT p.name, SUM(mt.price) AS total\_paid   
FROM Patient p  
JOIN Test\_Result tr ON p.Patient\_ID = tr.Patient\_ID   
JOIN Medical\_Test mt ON tr.Test\_ID = mt.Test\_ID   
WHERE p.Patient\_ID = 1001 AND tr.date >= DATE\_SUB(CURDATE(), INTERVAL 3 YEAR)   
GROUP BY p.Patient\_ID;

SELECT p.name, SUM(mt.price) AS total\_paid

FROM Patient p

JOIN Test\_Result tr ON p.Patient\_ID = tr.Patient\_ID

JOIN Medical\_Test mt ON tr.Test\_ID = mt.Test\_ID

WHERE p.Patient\_ID = 12527 AND tr.date >= DATE\_SUB(CURDATE(), INTERVAL 3 YEAR)

GROUP BY p.Patient\_ID;

**8. Bonus: Application UI**

made a desktop application using Python and Tkinter that works with the medical lab database I created.

* First, build a program that opens a window with tabs. Each tab shows one of the database tables like Laboratorian, Patient, Component, Medical Test, and Test Result.
* From the app, you can add, update, delete, and view data for each table.
* Made the app connect directly to the database and display the records in a table format.
* Also added a function to insert sample data the first time the app runs, so you can test the project easily.

**Explanation:**

This application makes it easier to manage the lab data without writing SQL commands every time.

**Steps to doing the app:**

1 - Use VScode to build and run the code

2 - Install Python Extension for VSCode

3- Creating a new folder

4- Create a new file named (medical\_lab\_app.py)

5- Write code to create the main window for the app using Tkinter

6- Create Tabs for Each Section

7- Create input forms where the user can add or update records, for each table

8- Create the SQLite Database for (Laboratorians, Patients, Medical Tests, Components, and Test Results)

9- Write SQL queries to create the tables and insert sample data

10- Write Python functions to handle the (Create, Read, Update, Delete) operations

11 - Link the (Add, Update, Delete) buttons in the UI to the functions that perform operations on the database

12- Run the App in VScode

**9. Explanation and Implementation**

In this project, we started by identifying the main entities needed for a medical laboratory system: Laboratorian, Patient, Medical Test, Test Result, Component, and an extra table called Test\_Component to handle the many-to-many relationship between tests and components.

Each entity was carefully designed with appropriate attributes. We used primary keys to uniquely identify each record and foreign keys to connect related tables. We also draw the ER diagram and show cardinality and participation constraints. This helped in planning how the tables would relate and ensured that the design supports all required operations.

**Implementation:**

After designing the schema, we wrote the SQL statements to create each table. We also added constraints like NOT NULL and PRIMARY KEY to protect data integrity.

Then, we inserted sample data (at least 10 records per table) to test the structure and relationships. This helped us run queries to check if the design worked as expected, such as finding patients who had specific tests or checking which components were below minimum quantity.

Finally, we created a Tkinter-based Python application. The app connects to the SQLite database and allows the user to add, update, delete, and view records for all entities through a graphical user interface.