Nile University

SCHOOL OF BIOTECHNOLOGY

DATABASE PROJECT

Medicine Inventory

(Project 1)

Under the supervision of:

Dr. Mohamed Mahmoud ElSayeh

Under the supervision of the teaching assistant:

Dr. Esraa Magdy Anwar

Prepared by:

Youseif Hussein Farghaly

221000443

Table of Contents

1. Introduction

1.1. About the project:

Contributing to the provision and facilitation of the sale of medicines is one of the most important things that can be available to patients, so this database is a model that fulfills this role. However, its pivotal role is to facilitate the management of the medicine inventory, as doctors write the medical prescription, then employees take the medicine out of the designated area, then the invoice is calculated and presented to the patient, and all of this is stored on a database like the one we have in our hands in this project.

1.2. Reasons for choosing the project:

This project was selected over others because of its importance in facilitating medication access for patients by providing all necessary resources. Therefore, this database is one of the ways we can contribute in this field. Furthermore, the simplicity and ease of this project, as its concept was very clear and easy to implement.

2. Data Modeling Using the Entity–Relationship (ER) Model

I started by drawing a diagram that shows the entities and relationships in the database, as shown in **Figure 1**:

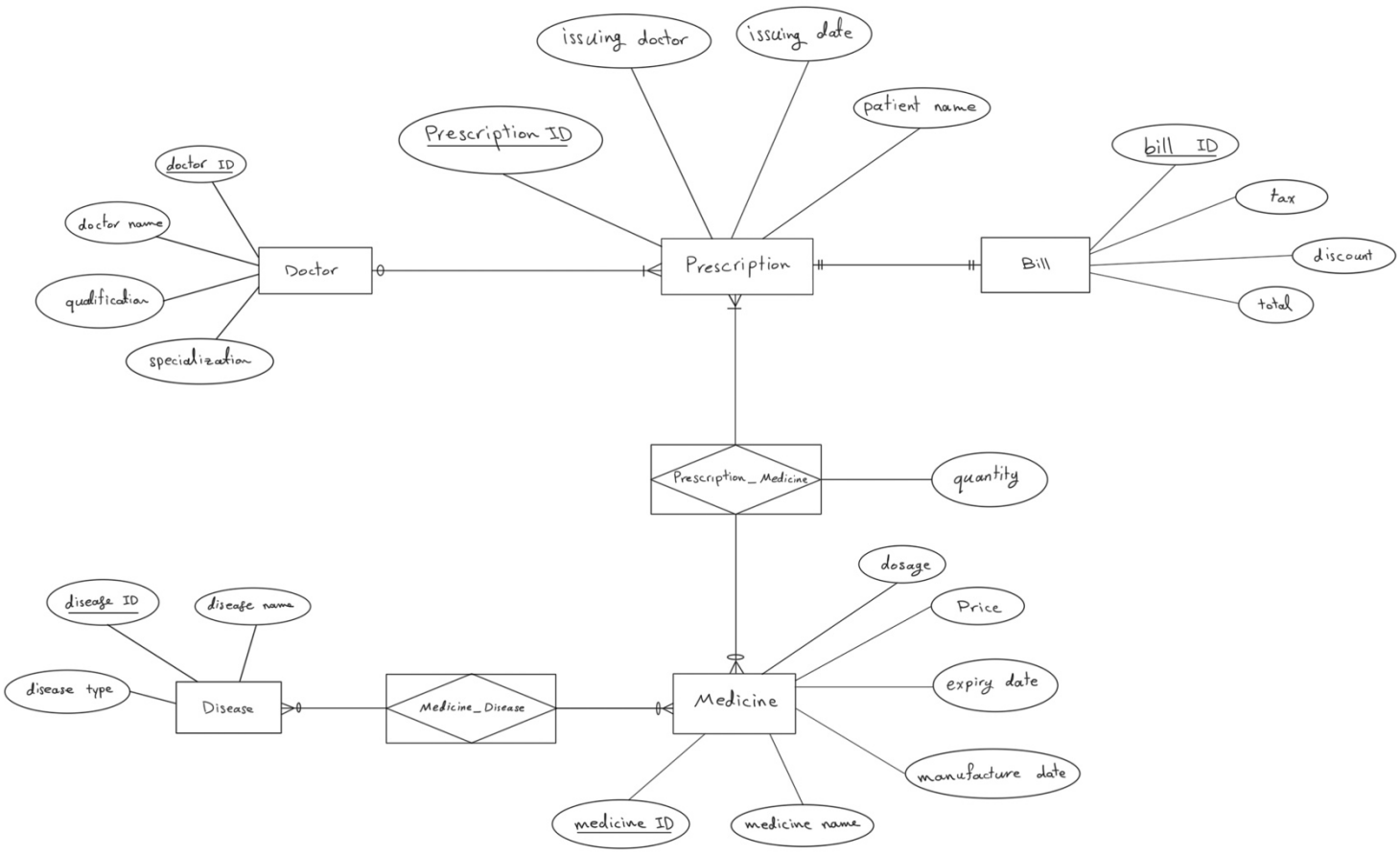

Figure 1

2.2. Relationships:

2.2.1. Doctor ↔ Prescription:

Relationship Type: (One-to-many)

Description: The doctor issues a prescription, and each doctor can issue multiple prescriptions, but each prescription is written by only one doctor. **Figure 2**.

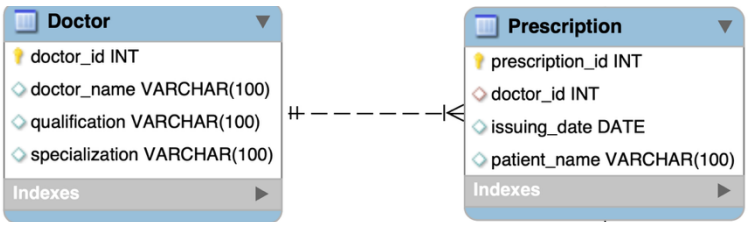

Figure 2

2.2.2. Prescription ↔ Prescription_Medicine ↔ Medicine:

Relationship Type: (Many-to-many) through an intermediate table "Prescription_Medicine".

Description: The prescription contains a specific combination in specific quantities. Each prescription contains a group of medications (or just one medication). Each medication can appear in more than one prescription. **Figure 3**.

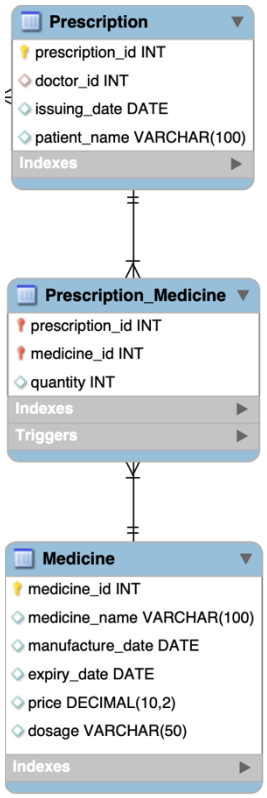

Figure 3

> ***Note:*** *An intermediate table is created because relational databases (such as: MySQL) do not support direct many-to-many relationships. Therefore, an intermediate table must be created. This not only allows us to represent the many-to-many relationship. Also, facilitates the process of adding or deleting any value. Furthermore, it prevents duplicate data in the same table.*

2.2.3. Medicine ↔ Medicine_Disease ↔ Disease:

Relationship Type: (Many-to-many) through an intermediate table "Medicine_Disease".

Description: Any medicine used to treat one or more diseases. **Figure 4**.

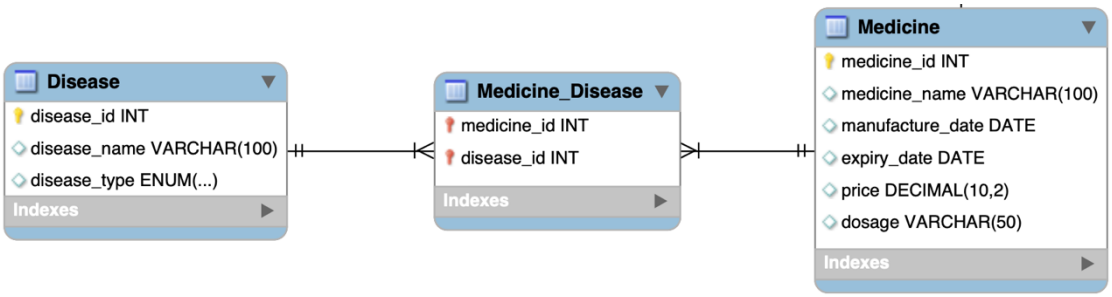

Figure 4

2.2.4. Prescription ↔ Bill:

Relationship Type: (One-to-One)

Description: Each prescription has a bill. In other words, each prescription has a bill calculated based on the medications contained in it.

Figure 5.

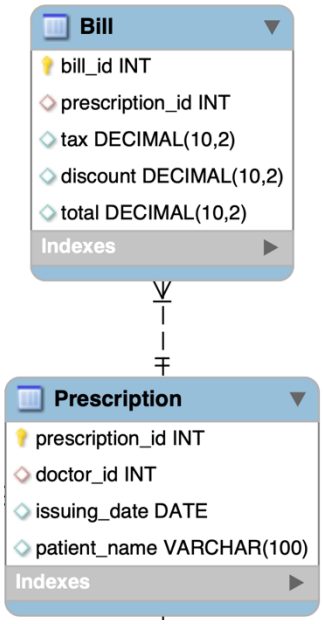

Figure 5

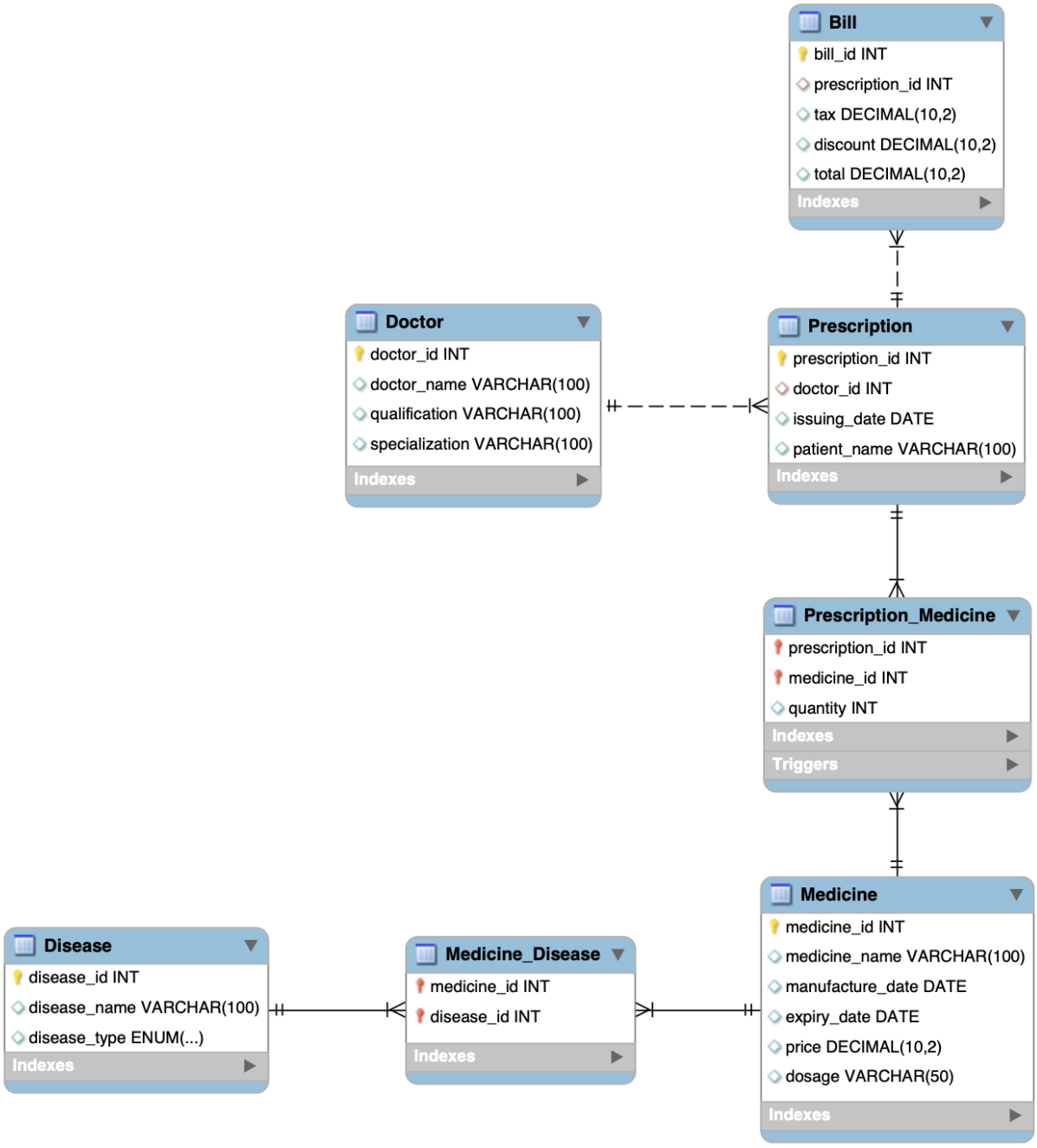

Figure 6: The full diagram shows the relationships extracted from MySQL.

3. Logical Design:

Mapping ERD converted to tables known as logical design as shown in **Figure 7**:

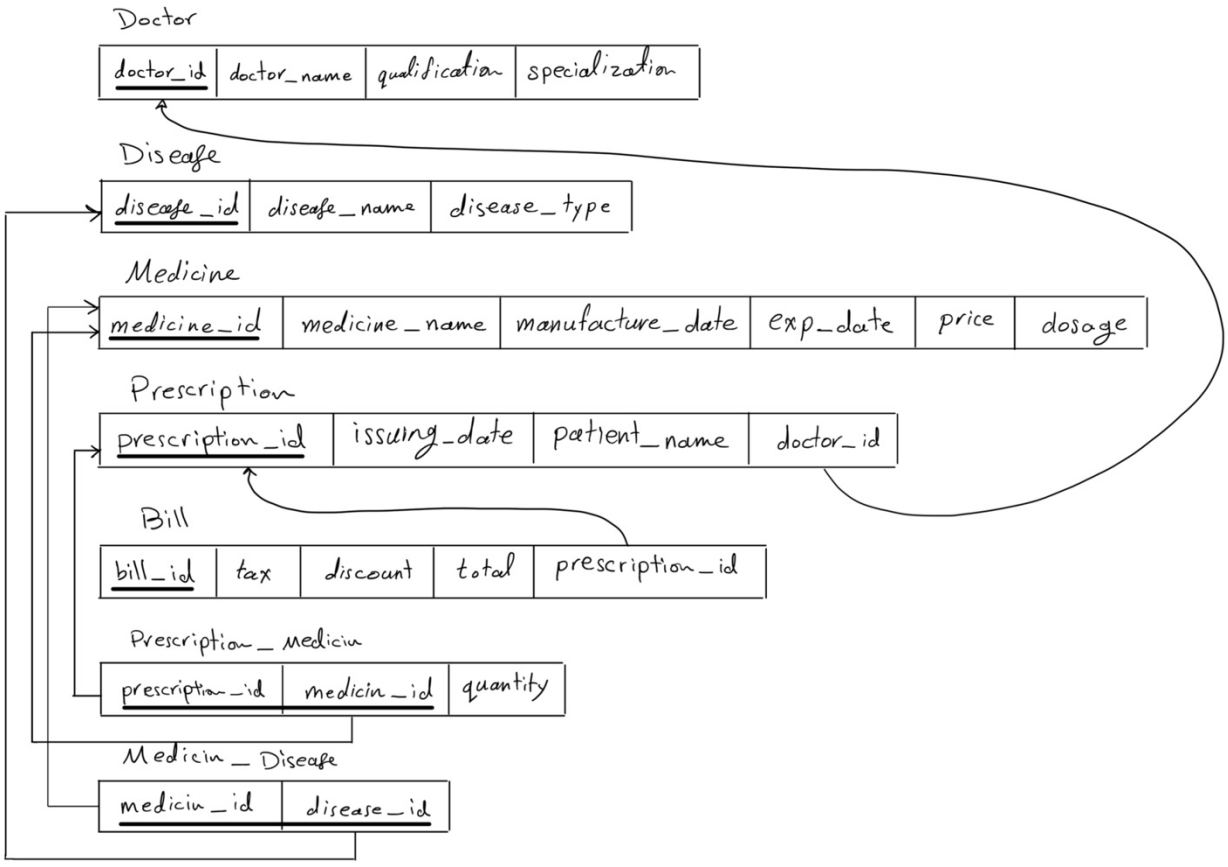

Figure 7

4. Create database:

Open the system > click on the "+" icon. (**Figure 8**)

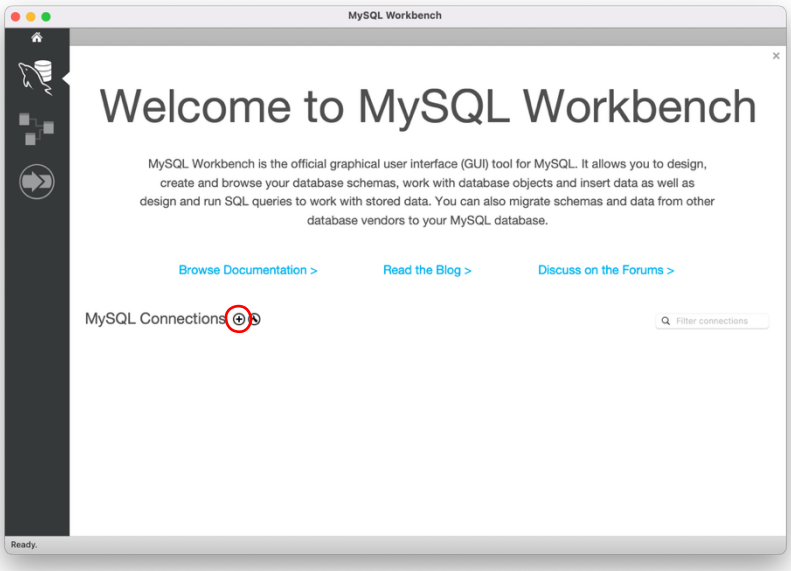

Figure 8

Specify the database name > Click "OK". (**Figure 9**). Now we able to start creating tables and coding using the SQL code.

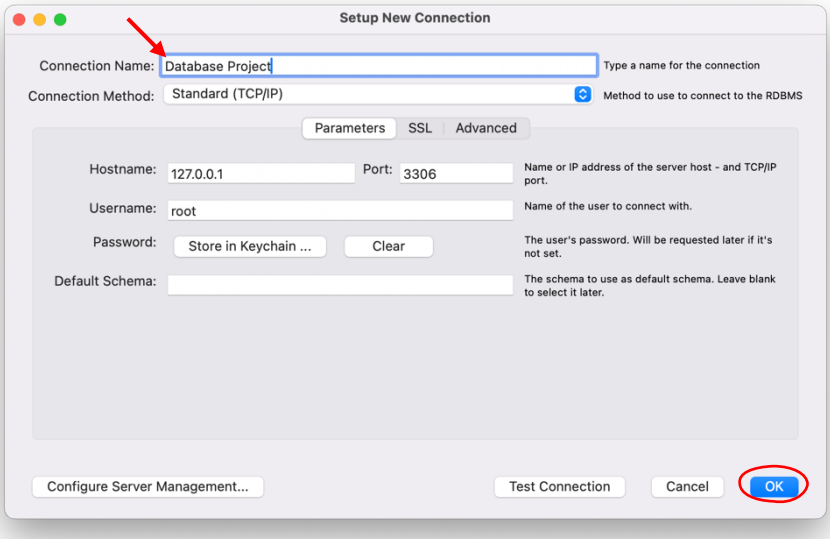

Figure 9

5. Writing the code:

5.1. defining the database:

```
1 •  CREATE DATABASE IF NOT EXISTS medicine_inventory;
2 •  USE medicine_inventory;
```

These two commands are used to create and specify the database I will be working with. If we do not use these lines, we will receive an error message (such as: No Database Selected) when we try to execute any command.

5.2. Create tables:

5.2.1. Table of doctors:

```
3 •  CREATE TABLE Doctor (
4        doctor_id INT PRIMARY KEY,
5        doctor_name VARCHAR(100),
6        qualification VARCHAR(100),
7        specialization VARCHAR(100)
8    );
```

It is an entity that represents the doctor who issues the prescription. For each doctor **Primary Key** (the primary identifier). It also has a specific name, qualification, and specialty.

5.2.2. Table of disease:

```
10  CREATE TABLE Disease (
11      disease_id INT PRIMARY KEY,
12      disease_name VARCHAR(100),
13      disease_type ENUM('infectious', 'deficiency', 'genetic hereditary', 'non-genetic hereditary')
14  );
```

Each disease has its own table, which contains the **Primary Key**, disease name, and disease type using ENUM (i.e. it only accepts specific values).

5.2.3. Table of Medicine:

```
16  CREATE TABLE Medicine (
17      medicine_id INT PRIMARY KEY,
18      medicine_name VARCHAR(100),
19      manufacture_date DATE,
20      expiry_date DATE,
21      price DECIMAL(10, 2),
22      dosage VARCHAR(50)
23  );
```

Each medication has a separate table containing: the **Primary Key**, the medication name, the manufacturing and expiration dates, the price, and finally the dosage.

```
25  CREATE TABLE Medicine_Disease (
26      medicine_id INT,
27      disease_id INT,
28      PRIMARY KEY (medicine_id, disease_id),
29      FOREIGN KEY (medicine_id) REFERENCES Medicine(medicine_id),
30      FOREIGN KEY (disease_id) REFERENCES Disease(disease_id)
31  );
```

An intermediate table is created between the disease table and the medication table (because the relationship is many-to-many).

The **Primary Key** of the Disease table and the Medication table are placed in this intermediate table, and code is written to specify the location of the Primary Key in each column, as shown in lines 29 and 30. In other words, I use an intermediate table that contains the primary keys from both tables as the **Composite Primary Key**.

The medication is used for more than one disease, and a disease may be treated with more than one medication.

5.2.4. Table of Prescription:

```
33  CREATE TABLE Prescription (
34      prescription_id INT PRIMARY KEY,
35      doctor_id INT,
36      issuing_date DATE,
37      patient_name VARCHAR(100),
38      FOREIGN KEY (doctor_id) REFERENCES Doctor(doctor_id)
39  );
```

Each prescription has a separate table containing a **Primary Key**, patient name, and date of issue. It also contains a **Primary Key** for each doctor, since each prescription is for only one doctor, but each doctor can issue multiple prescriptions. Therefore, the relationship between the doctor table and the prescription table is (one-to-many).

Line 39 is written to specify the location of the Primary Key, and to create the relationship between the two tables (one to many).

5.2.5. Table Prescription_ Medicine:

```
41  CREATE TABLE Prescription_Medicine (
42      prescription_id INT,
43      medicine_id INT,
44      quantity INT,
45      PRIMARY KEY (prescription_id, medicine_id),
46      FOREIGN KEY (prescription_id) REFERENCES Prescription(prescription_id),
47      FOREIGN KEY (medicine_id) REFERENCES Medicine(medicine_id)
48  );
```

Because each prescription may contain more than one drug, and each drug can be used in more than one recipe, I create an intermediate table between Prescription and Medicine. Furthermore, I add “quantity” as an additional attribute to this intermediate table. I also create a composite primary key. consists of the primary key for the prescription table and the primary key for the drug table.

5.2.6. Table of Bill:

```
50  CREATE TABLE Bill (
51      bill_id INT PRIMARY KEY,
52      prescription_id INT,
53      tax DECIMAL(10, 2),
54      discount DECIMAL(10, 2),
55      total DECIMAL(10, 2),
56      FOREIGN KEY (prescription_id) REFERENCES Prescription(prescription_id)
57  );
```

Each invoice is associated with a single prescription (one-to-one). This table contains taxes, discounts, and the final total. Additionally, it contains a primary key, in addition to a **Foreign Key**, which is the primary key for the prescription. Thus, I have established a relationship between the two tables.

6. Automatically generate a bill after entering a prescription medication:

```
DELIMITER //
CREATE TRIGGER update_bill_after_prescription
AFTER INSERT ON Prescription_Medicine
FOR EACH ROW
BEGIN
    DECLARE total_price DECIMAL(10,2);

    SELECT SUM(m.price * NEW.quantity)
    INTO total_price
    FROM Medicine m
    WHERE m.medicine_id = NEW.medicine_id;

    INSERT INTO Bill (bill_id, prescription_id, tax, discount, total)
    VALUES (NEW.prescription_id, NEW.prescription_id, 0.00, 0.00, total_price);
END;
//
DELIMITER ;
```

This is achieved by using a **Trigger**. It is a SQL code that is automatically executed when a specific event occurs (such as inserting, modifying, or deleting data in a table).

6.1. How a Trigger Works:

Determine when the trigger will run (before or after an operation). What type of operation triggers the trigger (i.e., Insert, or Update, or Delete). I also specify the table associated with the trigger.

In our case, in the database I created, the trigger name is set, as shown in line 60. Its runtime is set after a medication is entered into a prescription, as shown in line 61. Its function is to add the price of the medication multiplied by the quantity, then automatically enter the value into the Bill table.

From what has been mentioned, the importance of adding a trigger to a project like this is clear. Here is a table that illustrates the benefits of using a trigger (table 1):

Benefit	Clarification
Automation	It performs tasks automatically without manual intervention
Reduce errors	Do not need to remember to enter the invoice yourself
Maintain consistency	It intelligently and efficiently links tables together

Table 1

7. simple and complex queries:

I created three queries, as explained below:

7.1. List the name of doctors whose specialty is heart disease:

```
77 •    SELECT doctor_name
78      FROM Doctor
79      WHERE specialization LIKE '%heart disease%';
```

Using this query, could search for doctors whose specialty contains the term “heart disease”.

typed **SELECT**, from which I specify the data I want to display. Then, typed **FROM**, which specifies the table in which this data resides. Finally, typed **WARE**, which specifies the array containing this data. I then enter the data format, which, if present, will display the doctor's name.

7.2. List the deficiency diseases:

```
81 •    SELECT disease_name
82      FROM Disease
83      WHERE disease_type = 'deficiency';
```

runed a query with steps very similar to the previous query. However, the difference is in line 80, where I specify the type of deficiency disease. Therefore, deficiency diseases will be displayed.

7.3. List the most medicine sold in 2023:

```
85 •   SELECT m.medicine_name, SUM(pm.quantity) AS total_sold
86     FROM Medicine m
87     JOIN Prescription_Medicine pm ON m.medicine_id = pm.medicine_id
88     JOIN Prescription p ON p.prescription_id = pm.prescription_id
89     WHERE YEAR(p.issuing_date) = 2023
90     GROUP BY m.medicine_name
91     ORDER BY total_sold DESC
92     LIMIT 1;
```

With this query, we can find the most dispensed medication in 2023, based on the quantity of the medication in the prescriptions.

As shown in line 85, after typing SELECT, select the medication name from the Medicine table (the “M” symbol is an abbreviation). We calculate the total quantity dispensed for each medication, “SUM(pm.quantity)”. gived this sum a new name, so it appears in the results as "Total Sold."

After typing FROM, I define the table.

On line 87, I link the Medicine table to the Prescription_Medicine table, based on the matching of the “medicine_id” medication number,

since the Prescription_Medicine table is the intermediate table that identifies which medication is included in which prescription.

On line 88, I link the Prescription_Medicine table to the Prescription table based on the matching of the “prescription_id” prescription number, because we want to know the date of the prescription, so we know whether it was issued in 2023 or not.

The condition that appears on line 89 filters the results to include prescriptions issued in 2023. YEAR is a function that calculates the year from a date.

On line 90, I instruct the database to group data based on the drug's name, with each drug in a single row.

On line 91, I sort the results in descending order (from best-selling to least-selling).

Finally, on line 92, I select only the first result, the best-selling drug in 2023.

8. Filling the database:

Fill database tables with test data (10 records for each table). To do this, I used artificial intelligence to generate fake data. I entered it into the database using the following codes:

```
94 •  INSERT INTO Doctor (doctor_id, doctor_name, qualification, specialization) VALUES
95    (1, 'Dr. Ahmed Saeed', 'MBBS', 'Heart Disease'),
96    (2, 'Dr. Mona Hossam', 'MD', 'Diabetes'),
97    (3, 'Dr. Karim Nabil', 'PhD', 'Genetics'),
98    (4, 'Dr. Heba Adel', 'MBBS', 'Infectious Diseases'),
99    (5, 'Dr. Tamer Khaled', 'MD', 'Nutrition'),
100   (6, 'Dr. Rania Mostafa', 'PhD', 'Liver Disease'),
101   (7, 'Dr. Youssef Hani', 'MBBS', 'Eye Disorders'),
102   (8, 'Dr. Salma Farid', 'MD', 'Genetic Disorders'),
103   (9, 'Dr. Mahmoud Fathi', 'MBBS', 'Skin Diseases'),
104   (10, 'Dr. Eman Kamal', 'MD', 'Blood Pressure');
```

"INSERVE INTO was typed, select "Doctor" table, and enter the attributes within that table. then entered the data in the same order as the attributes entered, making sure whether the required data is a number or a string. repeat the process until we have completed the ten records.

The same codes were repeated (with different records) for the remaining tables.

```
INSERT INTO Disease (disease_id, disease_name, disease_type) VALUES
(1, 'Hypertension', 'non-genetic hereditary'),
(2, 'Diabetes', 'non-genetic hereditary'),
(3, 'Thalassemia', 'genetic hereditary'),
(4, 'COVID-19', 'infectious'),
(5, 'Scurvy', 'deficiency'),
(6, 'Night Blindness', 'deficiency'),
(7, 'Cystic Fibrosis', 'genetic hereditary'),
(8, 'Flu', 'infectious'),
(9, 'Rickets', 'deficiency'),
(10, 'Hepatitis B', 'infectious');

INSERT INTO Medicine (medicine_id, medicine_name, manufacture_date, expiry_date, price, dosage) VALUES
(1, 'Atenolol', '2023-01-10', '2025-01-10', 25.50, '50mg'),
(2, 'Insulin', '2022-05-15', '2024-11-15', 80.00, '10ml'),
(3, 'Vitamin C', '2023-03-20', '2025-03-20', 10.75, '500mg'),
(4, 'Tamiflu', '2023-02-11', '2024-02-11', 45.00, '75mg'),
(5, 'Genotropin', '2023-07-01', '2026-07-01', 150.00, '4IU'),
(6, 'Zincovit', '2023-08-05', '2025-08-05', 12.30, '1 tablet'),
(7, 'Paracetamol', '2023-06-10', '2025-06-10', 8.90, '500mg'),
(8, 'Penicillin', '2022-11-01', '2024-11-01', 35.00, '250mg'),
(9, 'Albendazole', '2023-10-15', '2025-10-15', 18.00, '400mg'),
(10, 'Hydroxyurea', '2023-12-25', '2026-12-25', 60.00, '500mg');
```

```
INSERT INTO Medicine_Disease (medicine_id, disease_id) VALUES
(1, 1),
(2, 2),
(3, 5),
(4, 8),
(5, 7),
(6, 9),
(7, 8),
(8, 4),
(9, 10),
(10, 3);
```

```
142 •  INSERT INTO Prescription (prescription_id, doctor_id, issuing_date, patient_name) VALUES
143    (1, 1, '2023-06-01', 'Ali Mohamed'),
144    (2, 2, '2023-06-03', 'Sara Yasser'),
145    (3, 3, '2023-06-05', 'Nour Khaled'),
146    (4, 4, '2023-06-06', 'Hassan Samir'),
147    (5, 5, '2023-06-07', 'Mona Hesham'),
       (6, 6, '2023-06-08', 'Karim Adel'),
149    (7, 7, '2023-06-09', 'Laila Ibrahim'),
150    (8, 8, '2023-06-10', 'Fady Emad'),
151    (9, 9, '2023-06-11', 'Doaa Mahmoud'),
152    (10, 10, '2023-06-12', 'Nader Hany');
153
154 •  INSERT INTO Prescription_Medicine (prescription_id, medicine_id, quantity) VALUES
155    (1, 1, 2),
156    (2, 2, 1),
157    (3, 3, 3),
158    (4, 4, 1),
       (5, 5, 1),
160    (6, 6, 2),
161    (7, 7, 3),
162    (8, 8, 2),
163    (9, 9, 1),
164    (10, 10, 2);
```

To be able to display the records write:

```
166 •  SELECT * FROM Doctor;
167 •  SELECT * FROM Disease;
168 •  SELECT * FROM Medicine;
169 •  SELECT * FROM Prescription;
170 •  SELECT * FROM Prescription_Medicine;
171 •  SELECT * FROM Bill;
```

9. Conclusion:

Serving the pharmaceutical sector is One of the noble things you can do for patients. Whatever means are taken to achieve this goal should be considered and not underestimated.

What MySQL has demonstrated is that this database works efficiently. It is a simple form to which more can be added, improved, and developed further. Perhaps this is just the beginning.