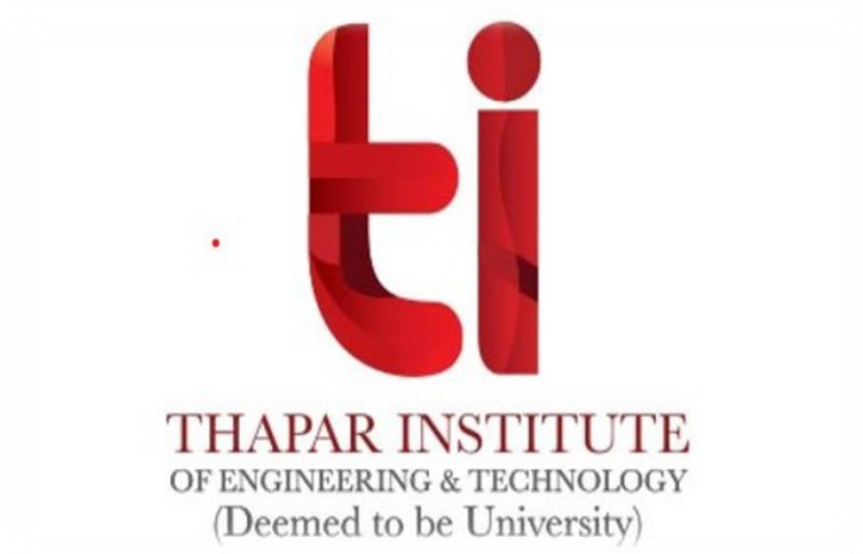
Pharmacy Management System

Course: Database Management System (UCS310)



Submitted to:
Ms. Neenu Garg

Submitted by:

Manya Singh	[102317119]
Arpita Sethi	[102317130]
Lavish	[102317132]
Prabhsimrat Singh	[102317135]

INDEX

Problem Statement

In the healthcare industry, pharmacies play a critical role in the distribution of medicines to patients. However, many small and mid-sized pharmacies still rely on manual methods for tracking inventory, generating bills, and managing customer records. These manual processes are prone to errors, time-consuming, and inefficient when handling large volumes of data.

This project aims to design and implement a **Pharmacy Management System** that provides a robust and scalable relational database solution to manage the core operations of a pharmacy.

The system will streamline the following processes:

- Recording and managing **medicine details**, including batch numbers, expiry dates, pricing, and quantity.
- Managing **user accounts** for pharmacists or system admins who access the database.
- Maintaining **customer profiles** and tracking their purchase histories.
- Generating **bills** for each transaction and tracking items per bill.
- Ensuring **inventory consistency** through automated updates and triggers.

Thus the need for an efficient, accurate, and scalable Pharmacy Management System is essential to reduce human error, maintain real-time inventory control, and streamline the billing and customer management processes. By designing a robust relational database backed by SQL and normalization principles, this project addresses the limitations of manual systems while laying the groundwork for digital transformation in small to medium-scale pharmacy operation.

Scope of the Project

This project focuses exclusively on the backend database design and logic of a single pharmacy unit. It includes the creation and management of tables related to medicines, users, customers, and bills, along with appropriate relationships between them. The system supports sample data insertion, SELECT queries, and execution snapshots, all built using MySQL. It does cover a front-end user interface and the schema is designed to be scalable for integration with such features in the future. The system ensures efficient and normalized data storage, aligning with academic and industrial best practices for DBMS applications.

Core CRUD Operations in the System

CRUD stands for Create, Read, Update, and Delete — the four essential operations used to manage data in any relational database system. These operations are implemented in the Pharmacy Management System to ensure complete control over the data lifecycle of medicines, customers, users, and billing records.

- **Create** – Add new data
  Example: Inserting a new medicine into the database.
- **Read** – Retrieve or view existing data
  Example: Viewing all available medicines or customer bills.
- **Update** – Modify existing data
  Example: Updating the quantity of a medicine after a sale.
- **Delete** – Remove data
  Example: Deleting an expired medicine from the inventory.

These operations reflect the real-world functionality of a pharmacy and are performed using SQL queries via phpMyAdmin or a connected Python interface.

ER DIAGRAM

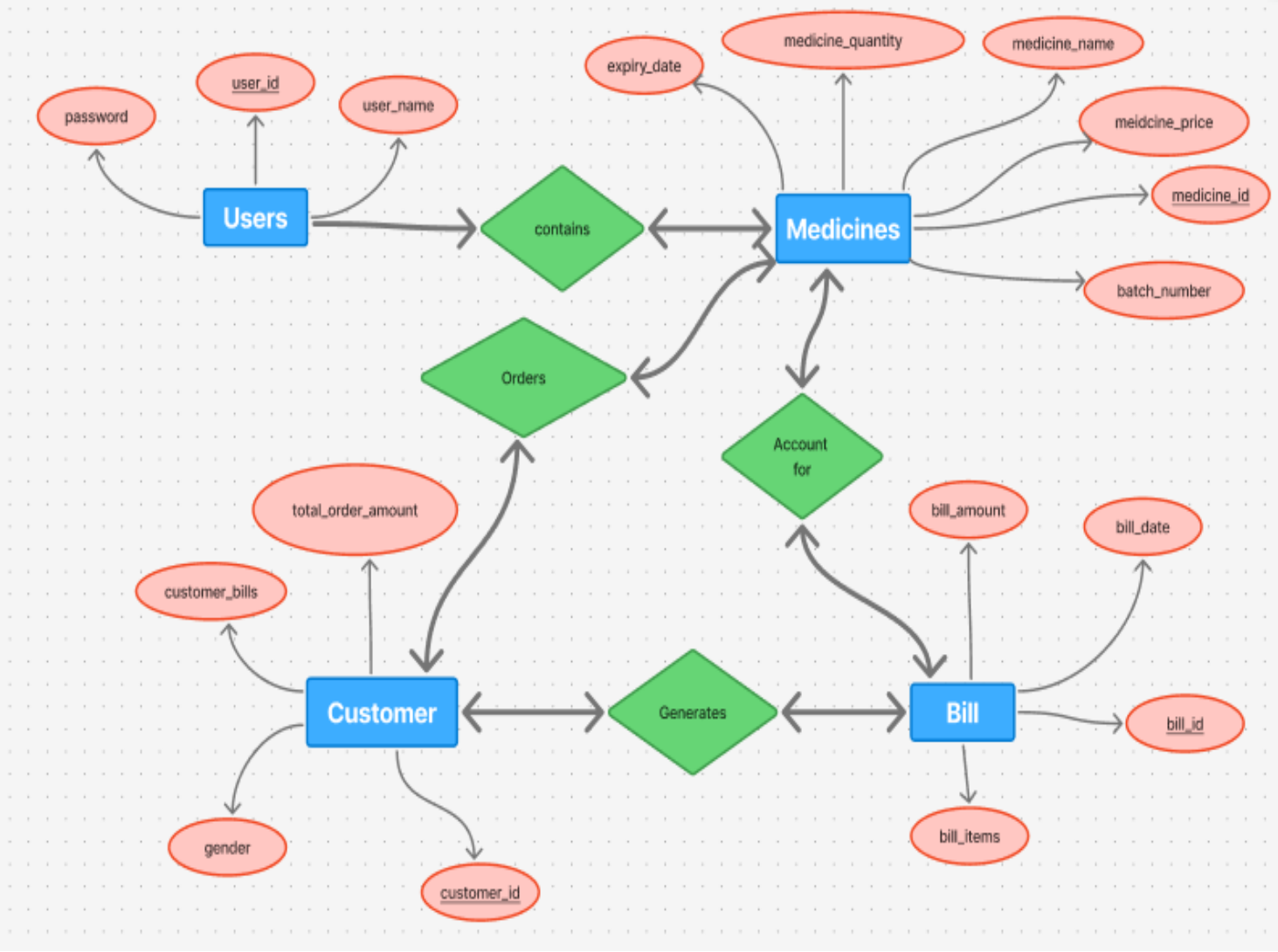
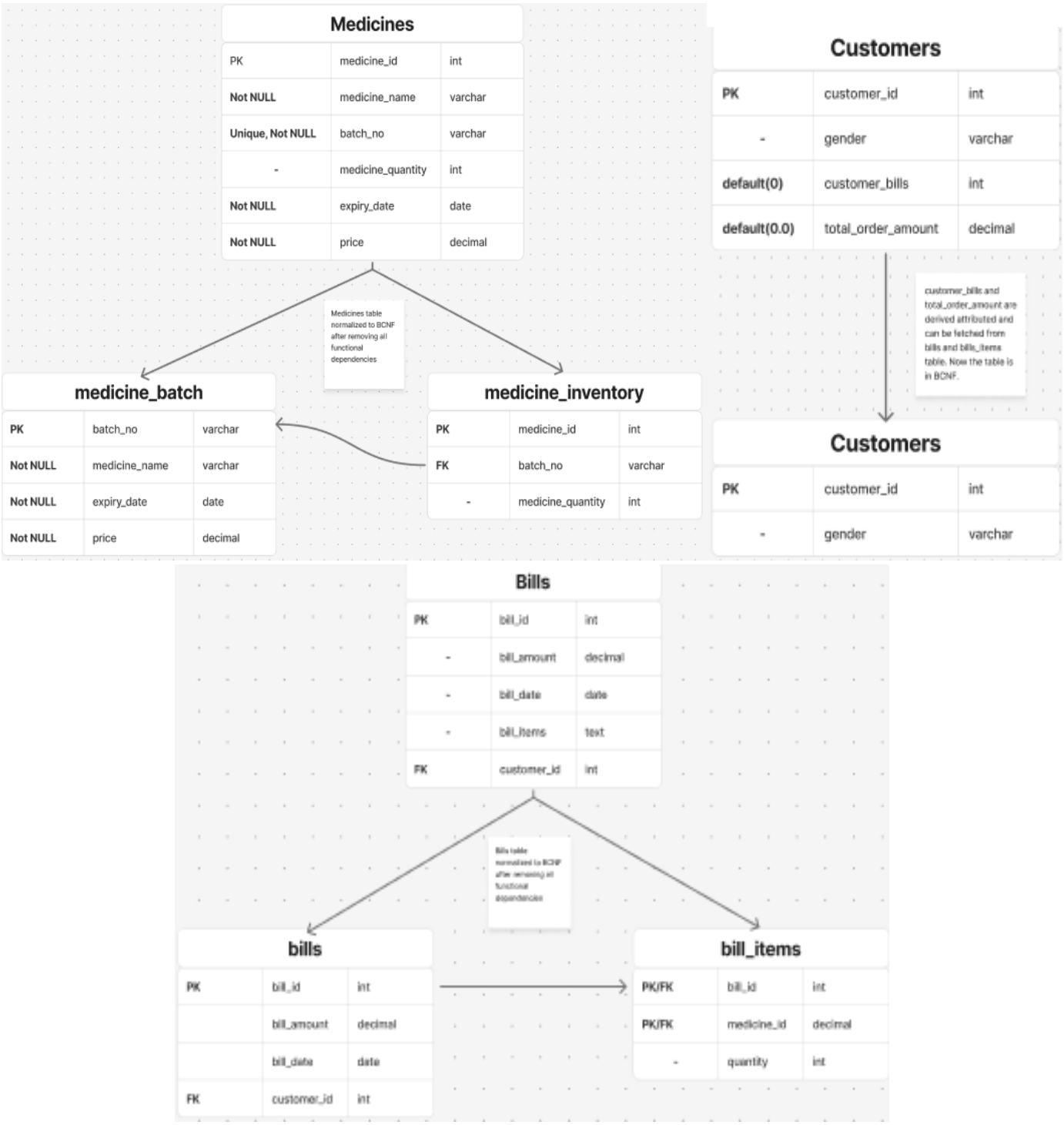


Fig1 Entity relationship diagram

This ER diagram includes four primary entities: Users, Medicines, Customers, and Bills. Each entity encompasses attributes that capture its role in the system. Specifically, the Users entity has attributes `user_id`, `user_name`, and `password` to represent system operators. The Medicines entity is defined by `medicine_id`, `medicine_name`, `batch_number`, `medicine_price`, `expiry_date`, and `medicine_quantity` to track each pharmaceutical item in inventory. The Customers entity contains `customer_id`, `name`, `gender`, `total_order_amount`, and a multivalued `customer_bills` attribute to store client data and the bills associated with that customer. The Bills entity includes `bill_id`, `bill_amount`, `bill_date`, and `bill_items` to record each sales transaction.

ER To Table


Medicines
PK
medicine_id
int
Not NULL
medicine_name
varchar
Unique, Not NULL
batch_no
varchar
-
medicine_quantity
int
Not NULL
expiry_date
date
Not NULL
price
decimal
Medicines table normalized to BCNF after removing all functional dependencies
medicine_batch
PK
batch_no
varchar
Not NULL
medicine_name
varchar
Not NULL
expiry_date
date
Not NULL
price
decimal
medicine_inventory
PK
medicine_id
int
FK
batch_no
varchar
-
medicine_quantity
int
Customers
PK
customer_id
int
-
gender
varchar
default(0)
customer_bills
int
default(0.0)
total_order_amount
decimal
customer_bills and total_order_amount are derived attributed and can be fetched from bills and bills_items table. Now the table is in BCNF.
Customers
PK
customer_id
int
-
gender
varchar
Bills
PK
bill_id
int
-
bill_amount
decimal
-
bill_date
date
-
bill_items
text
FK
customer_id
int
bills
PK
bill_id
int
bill_amount
decimal
bill_date
date
FK
customer_id
int
bill_items
PK/FK
bill_id
int
PK/FK
medicine_id
decimal
-
quantity
int

Users		
PK	user_id	int
Not NULL	user_name	varchar
Not NULL	password	varchar

Normalization

Normalization is applied to eliminate data redundancy, improve data integrity, and structure the database efficiently. In this project, all relations have been analyzed and transformed step-by-step into Boyce-Codd Normal Form (BCNF).

Below is a detailed breakdown of how each table has been normalized with justification.

1. **Users (Already in BCNF)**

Schema: user_id (PK), user_name, password

- Functional Dependencies:
  user_id → user_name, password
- Since user_id is a candidate key and the only determinant, the relation is already in BCNF.

2. **Medicines (Needs Decomposition)**

Schema: medicine_id (PK), medicine_name, batch_no, medicine_quantity, expiry_date, price

- Functional Dependencies:
  batch_no → medicine_name, expiry_date, price
  medicine_id → batch_no, medicine_quantity
- batch_no is a determinant but not a candidate key, violating BCNF.
→ Decomposition into:

A. Medicines_Batch: (batch_no PK, medicine_name, expiry_date, price)
B. Medicine_Inventory: (medicine_id PK, batch_no FK, medicine_quantity)

3. **Customers (Partially BCNF)**

Schema: customer_id (PK), gender, customer_bills, total_order_amount

- Functional Dependency: customer_id → gender, customer_bills, total_order_amount
- The table satisfies BCNF, but customer_bills and total_order_amount are derived values and can be calculated using simple SQL queries using aggregated functions like:

Total Order Amount:

```
SELECT customer_id, SUM(bill_amount) AS total_order_amount

FROM bills

GROUP BY customer_id;
```

Customer Bills:

```
SELECT customer_id, COUNT(*) AS customer_bills

FROM bills

GROUP BY customer_id;
```

4. **Bills (Requires Decomposition)**

Schema: bill_id (PK), bill_amount, bill_date, bill_items, customer_id (FK)

- bill_items is a multivalued attribute and violates 1NF.
→ Decomposition into:

A. Bills: (bill_id PK, bill_amount, bill_date, customer_id FK)
B. Bill_Items: (bill_id FK, medicine_id FK, quantity) with Composite PK (bill_id, medicine_id)

5. **Final Normalized Tables in BCNF**

1. Users: (user_id PK, user_name, password)
2. Medicines_Batch: (batch_no PK, medicine_name, expiry_date, price)
3. Medicine_Inventory: (medicine_id PK, batch_no FK, medicine_quantity)
4. Customers: (customer_id PK, gender)
5. Bills: (bill_id PK, bill_amount, bill_date, customer_id FK)
6. Bill_Items: (bill_id FK, medicine_id FK, quantity) with Composite PK (bill_id + medicine_id)

Generalization in Database Design

Generalization is a technique in database design used to reduce redundancy by combining common attributes of multiple entities into a higher-level entity. In this project, generalization

can be effectively applied to improve the design of the Users and Customers tables, as well as to further normalize medicine and billing data. This approach simplifies schema maintenance, enhances consistency, and supports scalability.

1.Generalizing Users and Customers as Person

Both Users and Customers share common characteristics such as name and gender, making them suitable for generalization into a single entity named Person. Role-based attributes are then defined in specialized child tables.

Generalized Schema:

```
CREATE TABLE people (
  person_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    gender VARCHAR(10),
    role ENUM('user', 'customer')
);
CREATE TABLE pharmacy_users (
    user_id INT PRIMARY KEY,
    username VARCHAR(50) UNIQUE,
    password VARCHAR(100),
    FOREIGN KEY (user_id) REFERENCES people(person_id)
);
CREATE TABLE pharmacy_customers (
    customer_id INT PRIMARY KEY,
    customer_bills INT DEFAULT 0,
    total_order_amount DECIMAL(10,2) DEFAULT 0.00,
    FOREIGN KEY (customer_id) REFERENCES people(person_id)
);
```

2. Separating Medicine Batches from Master Data

Medicines often have multiple batches with different expiry dates and prices. To reflect this real-world scenario, medicine details are separated into two tables: medicine_master and medicine_batch.

Normalized Schema:

```
CREATE TABLE medicine_master (
   medicine_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100)
);
CREATE TABLE medicine_batch (
   batch_id INT AUTO_INCREMENT PRIMARY KEY,
   medicine_id INT,
   batch_no VARCHAR(50),
   expiry_date DATE,
   price DECIMAL(10,2),
   quantity INT,
   FOREIGN KEY (medicine_id) REFERENCES medicine_master(medicine_id)
);
```

3. Decomposing Bill Items

To preserve 1NF and improve query flexibility, the multivalued bill_items text field should be replaced with a linking table that references each medicine associated with a bill.

Normalized Schema:

```
CREATE TABLE bill_items (
   bill_id INT,
   medicine_id INT,
   quantity INT,
   FOREIGN KEY (bill_id) REFERENCES pharmacy_bills(bill_id),
   FOREIGN KEY (medicine_id) REFERENCES pharmacy_medicines(id)
);
```

PL/SQL Snapshots

Procedures

```
CREATE OR REPLACE PROCEDURE update_medicine_quantity (
    p_id       IN NUMBER,
    p_quantity IN NUMBER
)
AS
BEGIN
    UPDATE medicines
    SET quantity = p_quantity
    WHERE id = p_id;
    COMMIT;
END;
/
update_medicine_quantity(p_id => 1, p_quantity => 300);
DBMS_OUTPUT.PUT_LINE('Medicine quantity updated.');
```

Output :

```
Medicine quantity updated.
```

	ID	NAME	BATCH_NO	EXPIRY_DATE	PRICE	QUANTITY
1	1	Paracetamol 500mg	B101	3/31/2026, 12:00:0(	15	300

Functions-

```
CREATE OR REPLACE FUNCTION get_total_sales(p_date IN DATE)
RETURN NUMBER
AS
    total_sales NUMBER;
BEGIN
    SELECT NVL(SUM(bill_amount), 0)
    INTO total_sales
    FROM bill
    WHERE TRUNC(bill_date) = TRUNC(p_date);
    return total_sales;
END;
/
    v_total_sales := get_total_sales(TO_DATE('2025-05-08', 'YYYY-MM-DD'));
    DBMS_OUTPUT.PUT_LINE('Total sales on 2025-05-08: ' || v_total_sales);
```

Output :

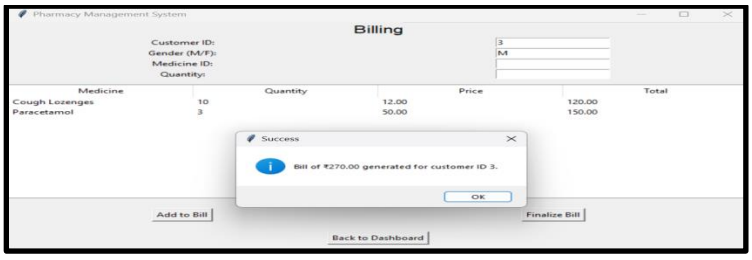
```
Total sales on 2025-05-08: 401
```

```
DECLARE
    v_total NUMBER := 0;
    v_subtotal NUMBER;
    v_message VARCHAR2(255);
    v_items VARCHAR2(4000) := '';
BEGIN
    add_to_bill(6, 5, v_subtotal, v_message);
    v_total := v_total + v_subtotal;
    v_items := v_items || 'Metformin 500mg x5, ';

    add_to_bill(5, 10, v_subtotal, v_message);
    v_total := v_total + v_subtotal;
    v_items := v_items || 'Cetirizine 10mg x10, ';

    v_items := RTRIM(v_items, ', ');
    finalize_bill(v_total, SYSDATE, v_items, 3);
END;
/
```

Output :

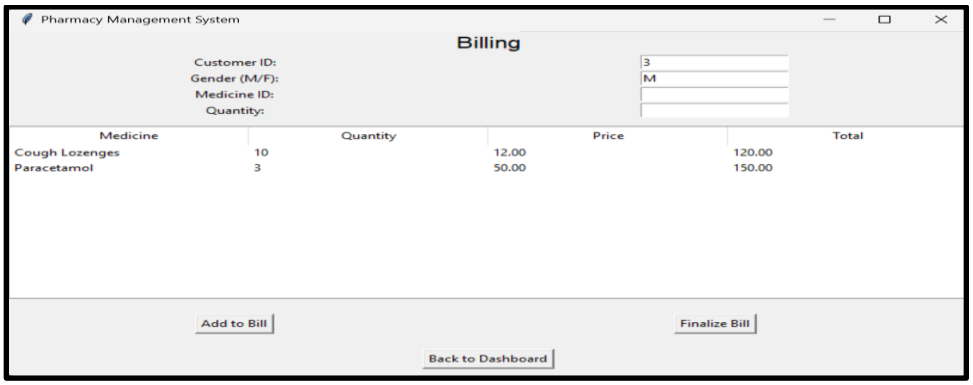


```
DECLARE
    v_total NUMBER := 0;
    v_subtotal NUMBER;
    v_message VARCHAR2(255);
    v_items VARCHAR2(4000) := '';
BEGIN
    add_to_bill(3, 4, v_subtotal, v_message);
    v_total := v_total + v_subtotal;
    v_items := v_items || 'Ibuprofen 400mg x4, ';

    add_to_bill(7, 2, v_subtotal, v_message);
    v_total := v_total + v_subtotal;
    v_items := v_items || 'Omeprazole 20mg x2, ';

    v_items := RTRIM(v_items, ', ');
    finalize_bill(v_total, SYSDATE, v_items, 2);
END;
/
```

Output :



SQL Query Demonstrations

```
select *from customer
```

Output:

	CUSTOMER_ID	GENDER	CUSTOMER_BILLS	TOTAL_ORDER_AM
1	2	M	1	130
2	3	F	1	180
3	4	M	1	91

```
select *from bill
```

Output:

	BILL_ID	BILL_AMOUNT	BILL_DATE	BILL_ITEMS	CUSTOMER_ID
1	1	130	5/8/2025, 1:41:49 P	Ibuprofen 400mg x4, Omeprazole 20mg x2	2
2	2	180	5/8/2025, 1:41:59 P	Metformin 500mg x5, Cetirizine 10mg x10	3
3	3	91	5/8/2025, 1:41:59 P	Amlodipine 5mg x3, Azithromycin 500mg x1	4

```
select * from medicines where quantity<20
```

Output :

	ID	NAME	BATCH_NO	EXPIRY_DATE	PRICE	QUANTITY
1	61	Bosentan 62.5mg	B108	11/15/2026, 12:00:(	250	12
2	62	Telotristat 250mg	B109	9/30/2026, 12:00:0(	300	8

```
SELECT customer_id, total_order_amount
INTO v_top_customer_id, v_top_order_value
FROM customer
WHERE total_order_amount = (
    SELECT MAX(total_order_amount) FROM customer
);
```

Output:

```
Top customer ID: 3, Total Order Amount: 180
```

```
select * from medicines where expiry_date < sysdate+30
```

Output:

	ID	NAME	BATCH_NO	EXPIRY_DATE	PRICE	QUANTITY
1	61	Bosentan 62.5mg	B108	11/15/2026, 12:00:(	250	12
2	62	Telotristat 250mg	B109	9/30/2026, 12:00:0(	300	8

Conclusion

The Pharmacy Management System project successfully demonstrates the design and implementation of a relational database-driven application using MySQL and Python. The implementation includes creating the required database schema (tables for users, medicines, customers, and bills) and populating them with representative sample data.

The Python application establishes a connection to the MySQL server (via the MySQL Connector/Python driver) and executes SQL commands to create tables, insert records, and query data. All basic CRUD (Create, Read, Update, Delete) operations were implemented: for example, new medicines can be added to inventory, existing records (such as stock levels or patient bills) can be updated, and entries can be deleted or retrieved through SQL queries embedded in the Python code. The notebook demonstrates example queries and outputs that verify correct functionality (e.g., listings of current medicines and summarized customer billing information).

The database design enforces data integrity through appropriate primary and foreign key constraints (for instance, each bill record references a customer), and normalization ensures minimal redundancy. An entity-relationship model was translated into the actual tables (as shown in the design diagrams) before implementation.

The project reinforces academic learning by applying SQL DDL/DML commands and Python database APIs in practice: students gained hands-on experience writing SQL statements, managing transactions, and handling query results within a programming environment. Finally, the completed system is functionally sound and ready for real-world use at a basic level. It meets the stated requirements for a small pharmacy by correctly managing inventory, customer data, and billing. While a graphical interface or additional security features could be layered on in the future, the core backend and logic are fully operational and illustrate how theoretical DBMS concepts are applied in a practical application.

Future improvements

 Inventory Alert System : Add automatic low-stock or expiry alerts in the UI, so users are notified when medicines are about to run out or expire.

 Search & Filter Features in the GUI: Add real-time search boxes and filters (by name, batch, expiry, etc.) to quickly access medicine or customer records.

 Generate and Export Bills as PDF: Integrate a Python PDF library (like reportlab or fpdf) to generate printed or downloadable customer bills.

GitHub Repository Link: https://github.com/prabhsimrat28/Pharmacy_Management

References

1. **Silberschatz, A., Korth, H. F., & Sudarshan, S.**
   Database System Concepts (7th Edition), McGraw-Hill Education, 2019.

2. **Elmasri, R., & Navathe, S. B.**
   Fundamentals of Database Systems (7th Edition), Pearson, 2016.

3. **MySQL Documentation https://dev.mysql.com/doc/**

4. **W3Schools – SQL Tutorial https://www.w3schools.com/sql/**

5. **GeeksforGeeks – DBMS Concepts and MySQL Examples**
   https://www.geeksforgeeks.org/dbms/

6. **Apache Friends – XAMPP Guide https://www.apachefriends.org/**

7. **phpMyAdmin Documentation https://docs.phpmyadmin.net/**

8. **Official Python MySQL Connector Documentation**
   https://dev.mysql.com/doc/connector-python/en/