Pharmacy Management System

A PROJECT REPORT

Submitted by

Gagan Chethan [Reg No: RA2211003011335]

Manish Kumar [Reg No: RA2211003011283]

Sivaram Vinod [Reg No: RA2211003011275]

Under the Guidance of

Dr. R. Jebakumar

Associate Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

DEPARTMENT OF COMPUTING TECHNOLOGIES
SCHOOL OF COMPUTING
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this project report "**Pharmacy Management System**" is the bonafide work of **Gagan Chethan [Reg. No. RA2211003011335], Manish Kumar [Reg. No. RA2211003011283] and Sivaram Vinod [Reg. No. RA2211003011275]** who carried out the project work under my supervision in the project course **Database Management Systems [21CSC205P]** for the academic year 2023-2024 [II year / IV semester].

Date: 06.05.2024

Faculty in Charge
Dr. R. Jebakumar
Associate Professor
Department of Computing Technologies
SRMSIT -KTR

HEAD OF THE DEPARTMENT
Dr. M. Pushpalatha
Professor
Department of Computing Technologies
SRMIST - KTR

ACKNOWLEDGEMENTS

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professor Dr. C. Lakshmi, Professor, Department of Computational Intelligence** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our my Course Faculty **Dr. R. Jebakumar, Associate Professor, Department of Computing Technologies,** for his assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr. M. Pushpalatha, Professor, Department of Computing Technologies** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

Gagan Chethan [RA2211003011335]

Manish Kumar [RA2211003011283]

Sivaram Vinod [RA2211003011275]

ABSTRACT

A Pharmacy Management System (PMS) is a comprehensive software solution designed to streamline the operations of a pharmacy, enhancing efficiency, accuracy, and customer service. The system typically encompasses various modules such as inventory management, prescription processing, patient profiles, billing, and reporting. By automating these critical tasks, a PMS reduces the likelihood of errors in prescription processing, ensures better inventory control, and facilitates seamless communication between pharmacists, physicians, and patients. Moreover, it offers functionalities like medication interaction checks and dosage monitoring, improving patient safety and adherence to treatment plans. Additionally, a PMS often integrates with electronic health record (EHR) systems, allowing for seamless exchange of patient information and enhancing the continuity of care across healthcare settings.

The implementation of a Pharmacy Management System not only enhances operational efficiency but also brings about significant benefits in terms of compliance, accountability, and customer satisfaction. Through features like electronic prescribing and automated refill reminders, it simplifies the prescription fulfillment process, reducing wait times for patients and improving overall service quality. Furthermore, by generating comprehensive reports on sales, inventory levels, and medication usage patterns, a PMS enables pharmacists to make data-driven decisions, optimize inventory stocking, and identify opportunities for cost savings. Overall, the adoption of a robust Pharmacy Management System represents a pivotal step for modern pharmacies in delivering high-quality healthcare services while staying competitive in today's rapidly evolving healthcare landscape.

TABLE OF CONTENTS

LIST OF FIGURES

1 PROBLEM STATEMENT

The current landscape of pharmacy management faces several challenges, necessitating the development of effective solutions. One key issue revolves around manual and disjointed processes, leading to inefficiencies in inventory management, prescription fulfillment, and patient care. Furthermore, the lack of integration between disparate systems often results in data discrepancies and communication gaps, hindering the delivery of timely and accurate pharmaceutical services. Additionally, regulatory compliance requirements continue to evolve, posing a constant challenge for pharmacies to adapt and maintain adherence. These challenges underscore the critical need for a robust pharmacy management system that can automate tasks, streamline workflows, enhance communication, and ensure compliance, ultimately improving the overall efficiency and quality of pharmacy operations.

2 Problem understanding and ER Model

2.1 Identification of Entity and Relationships

Entities:

Patient: Represents individuals receiving medications from the pharmacy. Attributes may include patient ID, name, contact information, and medical history.

Pharmacist: Represents pharmacists working within the pharmacy. Attributes may include pharmacist ID, name, contact information, and credentials. Physician: Represents doctors who prescribe medications to patients. Attributes may include physician ID, name, contact information, and specialty.

Medication: Represents individual drugs stocked and dispensed by the pharmacy. Attributes may include medication ID, name, dosage, manufacturer, and expiration date.

Prescription: Represents a doctor's authorization for a patient to receive a specific medication. Attributes may include prescription ID, patient ID, physician ID, medication ID, dosage instructions, and issue date.

Inventory: Represents the stock of medications within the pharmacy. Attributes may include medication ID, quantity on hand, reorder level, and unit cost. Transaction: Represents the act of dispensing medication to a patient. Attributes may include transaction ID, prescription ID, pharmacist ID, patient ID, medication ID, quantity dispensed, and transaction date.

Relationships:

Patient-Prescription: A patient can have multiple prescriptions, but each prescription is associated with only one patient.

Physician-Prescription: A physician can issue multiple prescriptions, but each prescription is issued by only one physician.

Medication-Inventory: Each medication has a corresponding inventory record to track its stock level and other inventory-related information.

Medication-Prescription: Each prescription is associated with a specific medication that the patient needs to receive.

Prescription-Transaction: Each prescription results in one or more transactions when the medication is dispensed to the patient.

Pharmacist-Transaction: Each transaction is handled by a pharmacist, and a pharmacist can handle multiple transactions.

Patient-Transaction: Each transaction involves a patient receiving medication, and a patient can have multiple transactions over time.

2.2 ER Model for Pharmacy Management System

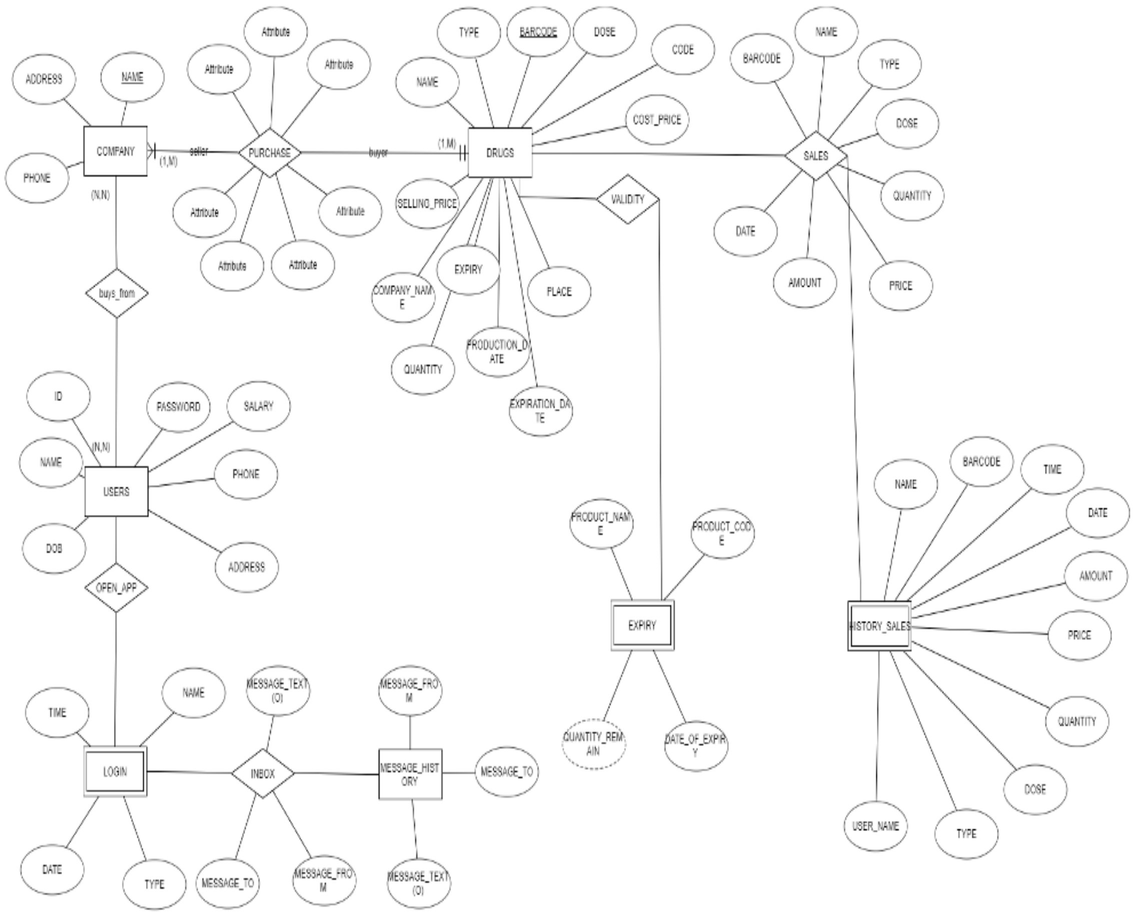


Fig. 2.1

In the Pharmacy Management System's Entity-Relationship (ER) model, key entities include Company, Drugs, Sales, Users, and History Sales. The Company entity represents pharmaceutical manufacturers, while Drugs encompass available pharmaceutical products. Sales records transactions, Users denote system users, and History Sales stores past sales records. Relationships in the model define interactions between these entities. The Manufactures relationship links Company to Drugs, indicating which companies produce which drugs. Performs associates Users with Sales, detailing who conducts each transaction. Contains connects Sales to Drugs, specifying which drugs are sold in each transaction. Logs In establishes the relationship between Users and the login system, ensuring secure access. Lastly, Logs Sales connects Sales to History Sales, maintaining a record of all sales transactions over time. This ER model succinctly organizes data and relationships within the Pharmacy Management System, facilitating efficient system operation and management.

3 Design of Relational Schemas

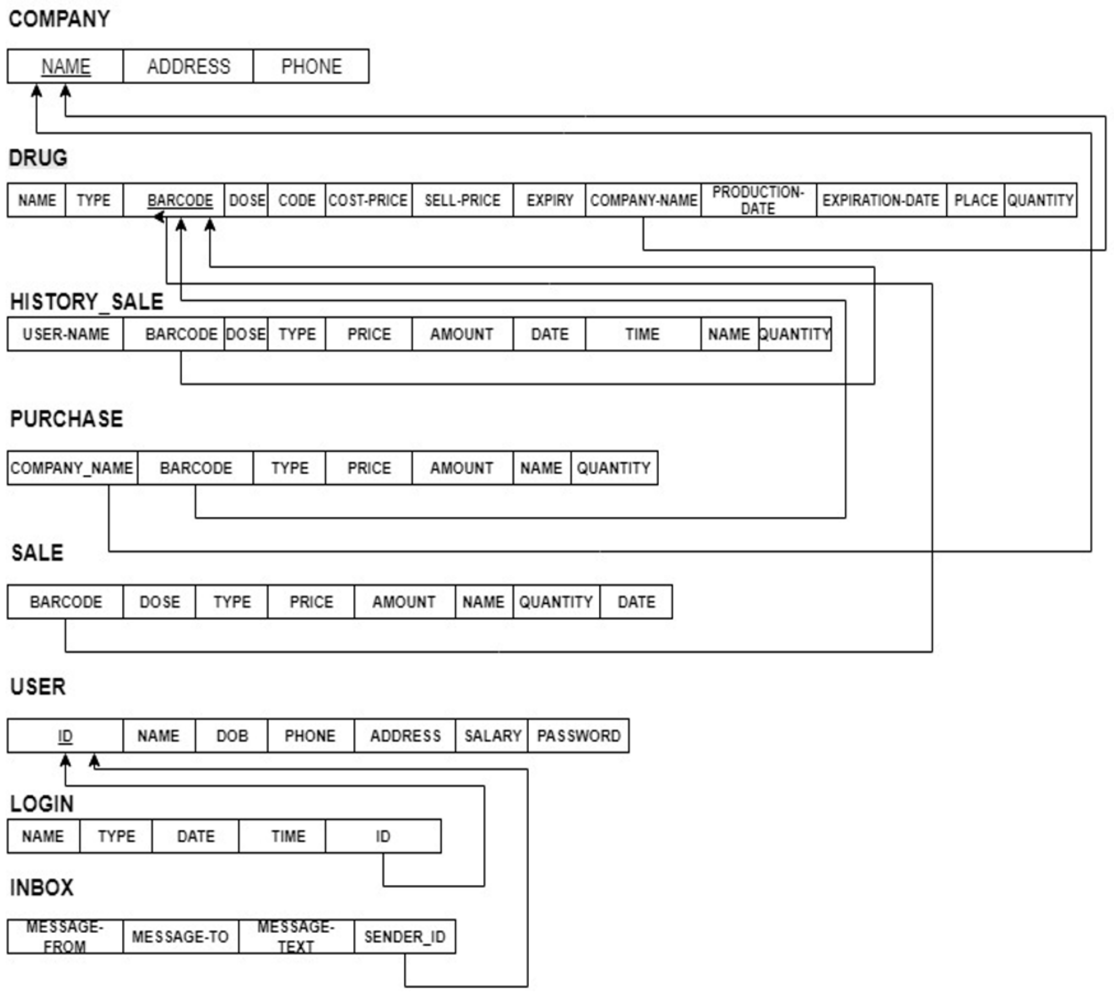
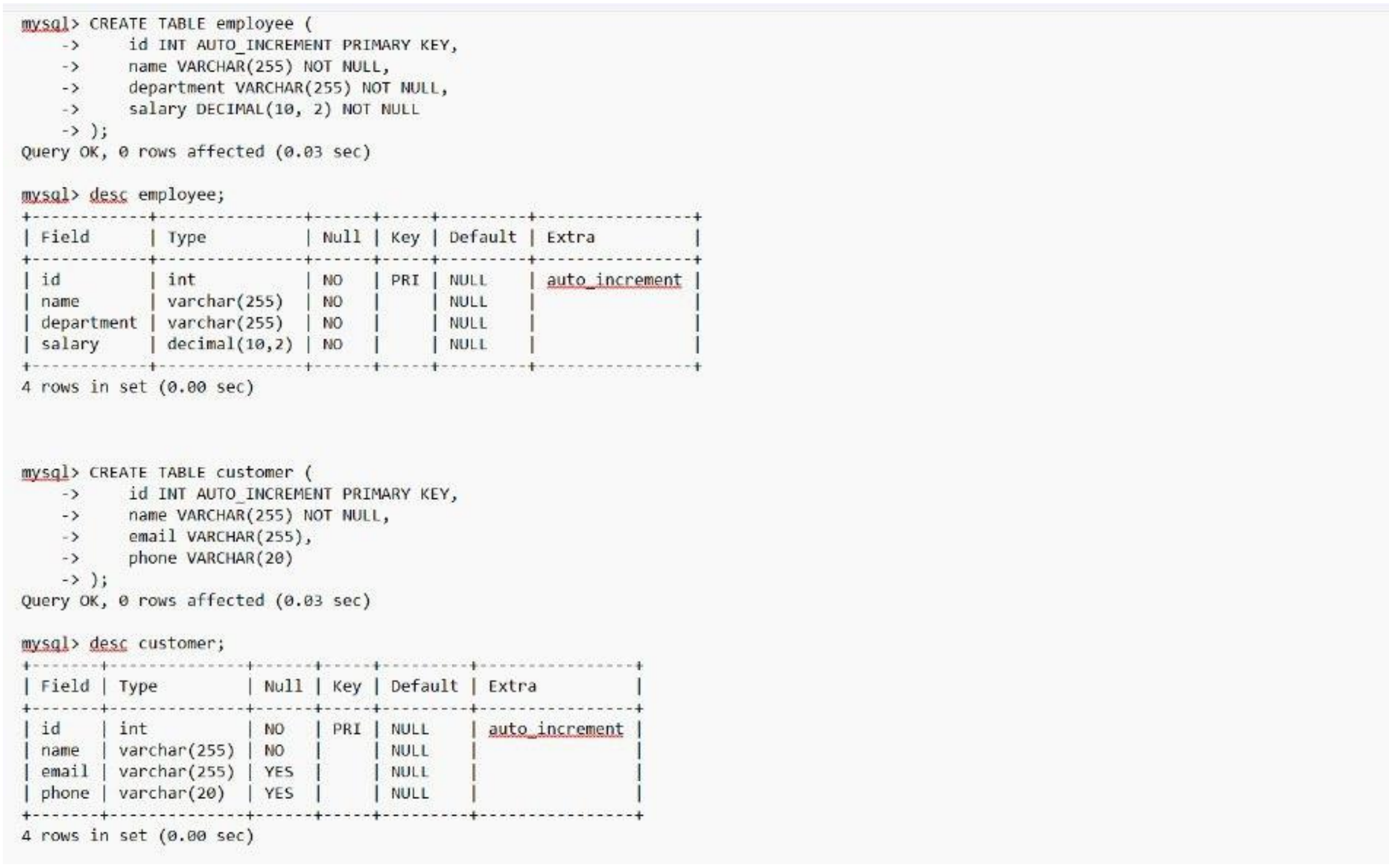


Fig. 3.1

In the relational schema for the Pharmacy Management System, each entity is represented as a table, with attributes defining their properties. The Company table includes attributes such as Company_ID, Name, Address, and Contact Information, with a Manufacturer_ID foreign key linking to the Drugs table. The Drugs table, in turn, contains attributes like Drug_ID, Name, Expiry Date, Price, and Quantity in Stock, with a Manufacturer_ID foreign key linking back to the Company table. The Sales table records transactions, with attributes including Sale_ID, Date, Time, and Total Price, and a User_ID foreign key linking to the Users table, which stores information such as User_ID, Name, Role, Username, and Password. Additionally, the History Sales table mirrors the Sales table but retains records of past transactions. Relationships are established through foreign keys, such as the Company_ID and Manufacturer_ID linking Companies to Drugs, the User_ID linking Users to Sales, and the Sale_ID connecting Sales to History Sales. This relational schema forms a structured framework for organizing and accessing data within the Pharmacy Management System, ensuring data integrity and facilitating efficient data manipulation and retrieval.

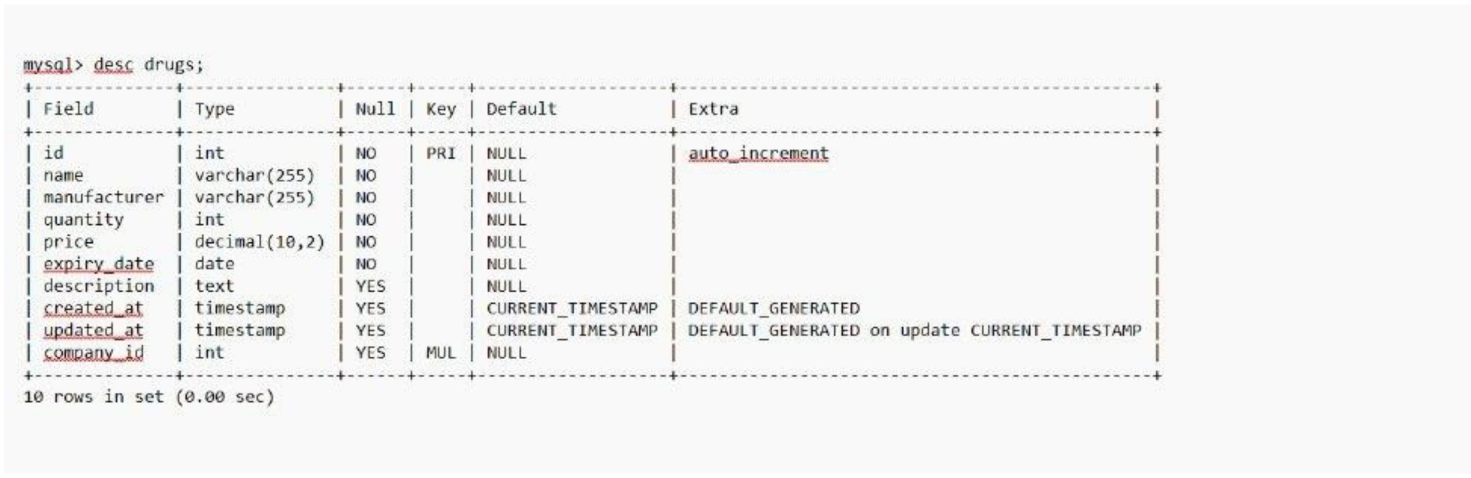
4 Creation of Database Tables



```
mysql> CREATE TABLE employee (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(255) NOT NULL,
    ->     department VARCHAR(255) NOT NULL,
    ->     salary DECIMAL(10, 2) NOT NULL
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> desc employee;
+------------+---------------+------+-----+---------+----------------+
| Field      | Type          | Null | Key | Default | Extra          |
+------------+---------------+------+-----+---------+----------------+
| id         | int           | NO   | PRI | NULL    | auto_increment |
| name       | varchar(255)  | NO   |     | NULL    |                |
| department | varchar(255)  | NO   |     | NULL    |                |
| salary     | decimal(10,2) | NO   |     | NULL    |                |
+------------+---------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)


mysql> CREATE TABLE customer (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(255) NOT NULL,
    ->     email VARCHAR(255),
    ->     phone VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> desc customer;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| id    | int          | NO   | PRI | NULL    | auto_increment |
| name  | varchar(255) | NO   |     | NULL    |                |
| email | varchar(255) | YES  |     | NULL    |                |
| phone | varchar(20)  | YES  |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Fig. 4.1

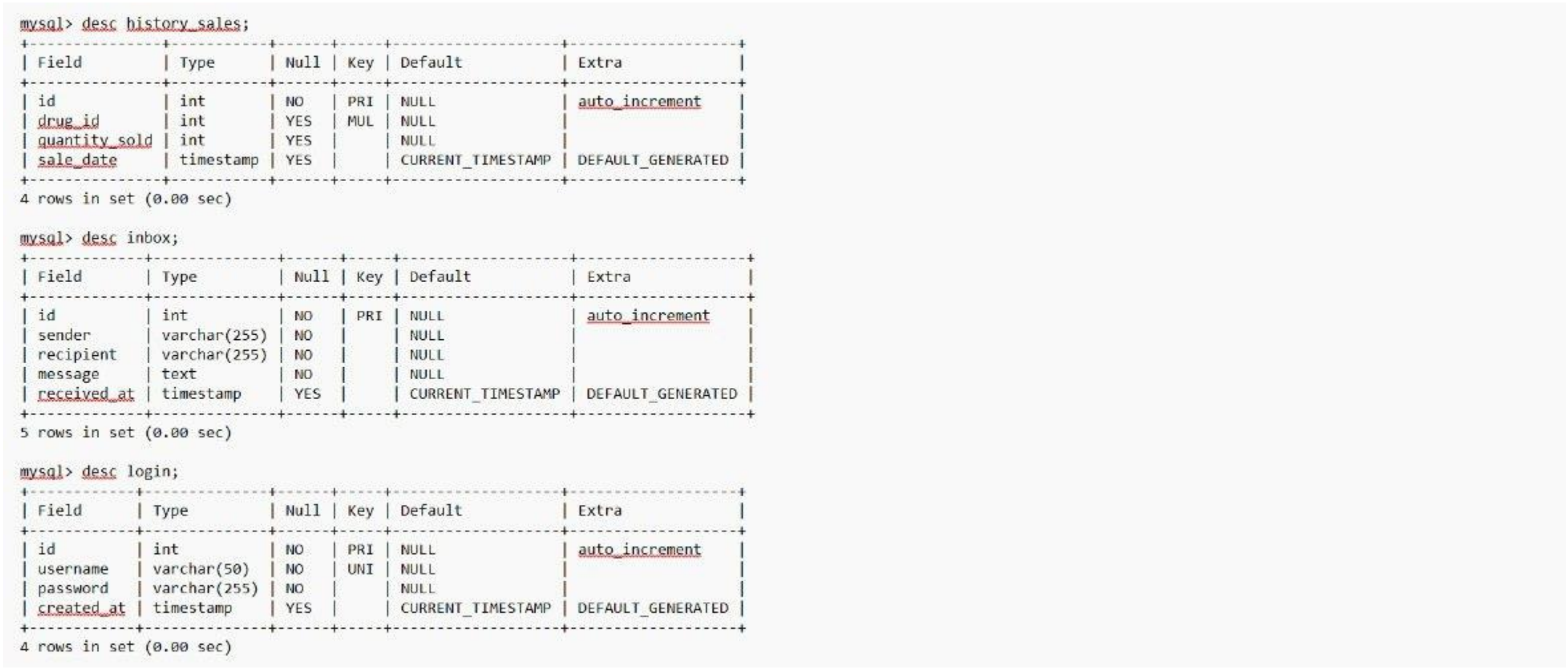
These two tables provide information about employees and customers and their connections. We begin by defining a table for employees, capturing details such as EmployeeID, FirstName, LastName, Email, Phone, Department, and Position. Simultaneously, the customer table holds fields like CustomerID, FirstName, LastName, Email, Phone, and Address, storing relevant customer information. While initially distinct, connections between employees and customers can be established through additional tables or by incorporating foreign keys within these tables to signify relationships. For instance, employees might manage specific customers, or customers could be linked to particular sales transactions overseen by employees. This structured approach facilitates efficient data management and retrieval within the database, enabling seamless operations in contexts like customer service and sales management.



```
mysql> desc drugs;
+--------------+---------------+------+-----+-------------------+-----------------------------------------------+
| Field        | Type          | Null | Key | Default           | Extra                                         |
+--------------+---------------+------+-----+-------------------+-----------------------------------------------+
| id           | int           | NO   | PRI | NULL              | auto_increment                                |
| name         | varchar(255)  | NO   |     | NULL              |                                               |
| manufacturer | varchar(255)  | NO   |     | NULL              |                                               |
| quantity     | int           | NO   |     | NULL              |                                               |
| price        | decimal(10,2) | NO   |     | NULL              |                                               |
| expiry_date  | date          | NO   |     | NULL              |                                               |
| description  | text          | YES  |     | NULL              |                                               |
| created_at   | timestamp     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED                             |
| updated_at   | timestamp     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTAMP |
| company_id   | int           | YES  | MUL | NULL              |                                               |
+--------------+---------------+------+-----+-------------------+-----------------------------------------------+
10 rows in set (0.00 sec)
```

Fig. 4.2

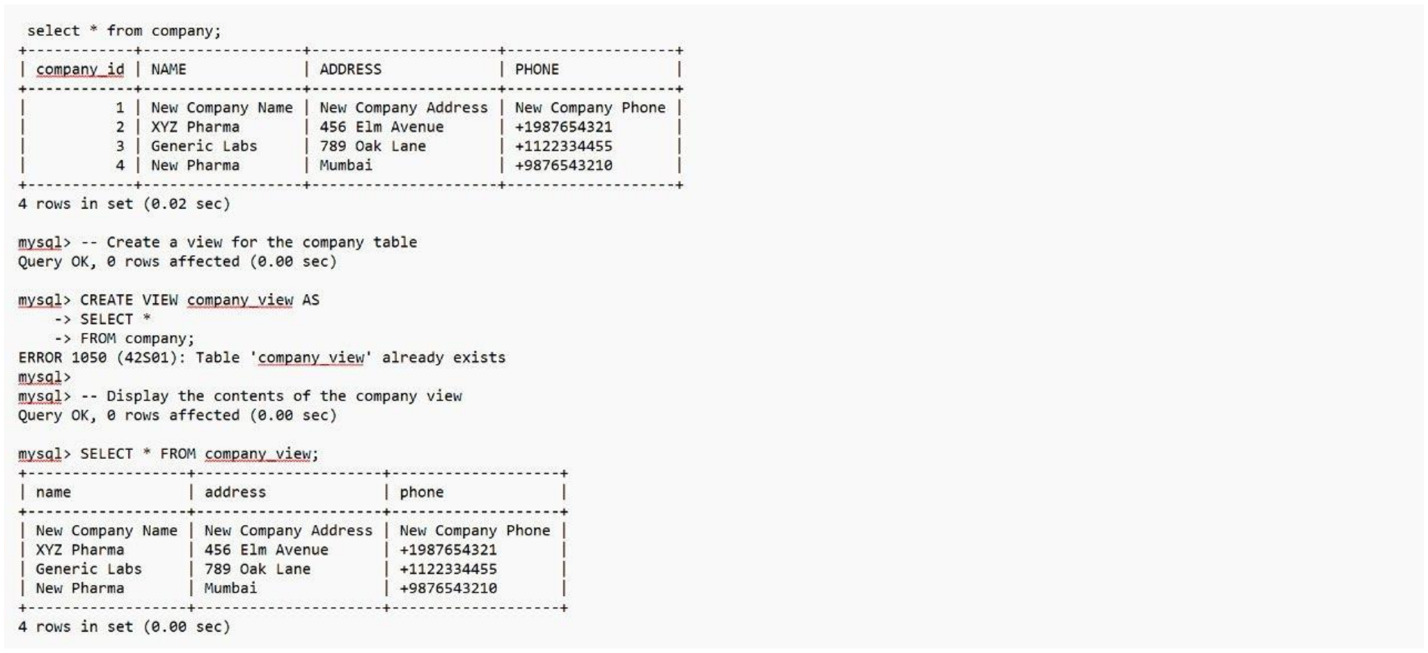
These tables provide information about drugs and their connections. We initiate the schema with a table for drugs, encompassing attributes such as DrugID, Name, Manufacturer, Expiry Date, Price, and Quantity in Stock. Concurrently, additional tables or foreign keys within the drug table can establish connections, representing relationships with entities like pharmaceutical companies, sales transactions, or historical data. For example, a "Manufacturer" field in the drug table can link to a separate table containing details about pharmaceutical companies, indicating which company produces each drug. Likewise, connections to sales transactions or historical records can track the sale of specific drugs over time, facilitating inventory management and sales analysis within the system. This structured approach ensures efficient organization and retrieval of drug-related information, supporting seamless operations within the pharmacy management system.



```
mysql> desc history_sales;
+---------------+-----------+------+-----+-------------------+-------------------+
| Field         | Type      | Null | Key | Default           | Extra             |
+---------------+-----------+------+-----+-------------------+-------------------+
| id            | int       | NO   | PRI | NULL              | auto_increment    |
| drug_id       | int       | YES  | MUL | NULL              |                   |
| quantity_sold | int       | YES  |     | NULL              |                   |
| sale_date     | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+---------------+-----------+------+-----+-------------------+-------------------+
4 rows in set (0.00 sec)

mysql> desc inbox;
+-------------+--------------+------+-----+-------------------+-------------------+
| Field       | Type         | Null | Key | Default           | Extra             |
+-------------+--------------+------+-----+-------------------+-------------------+
| id          | int          | NO   | PRI | NULL              | auto_increment    |
| sender      | varchar(255) | NO   |     | NULL              |                   |
| recipient   | varchar(255) | NO   |     | NULL              |                   |
| message     | text         | NO   |     | NULL              |                   |
| received_at | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-------------+--------------+------+-----+-------------------+-------------------+
5 rows in set (0.00 sec)

mysql> desc login;
+------------+--------------+------+-----+-------------------+-------------------+
| Field      | Type         | Null | Key | Default           | Extra             |
+------------+--------------+------+-----+-------------------+-------------------+
| id         | int          | NO   | PRI | NULL              | auto_increment    |
| username   | varchar(50)  | NO   | UNI | NULL              |                   |
| password   | varchar(255) | NO   |     | NULL              |                   |
| created_at | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+------------+--------------+------+-----+-------------------+-------------------+
4 rows in set (0.00 sec)
```

Fig. 4.3

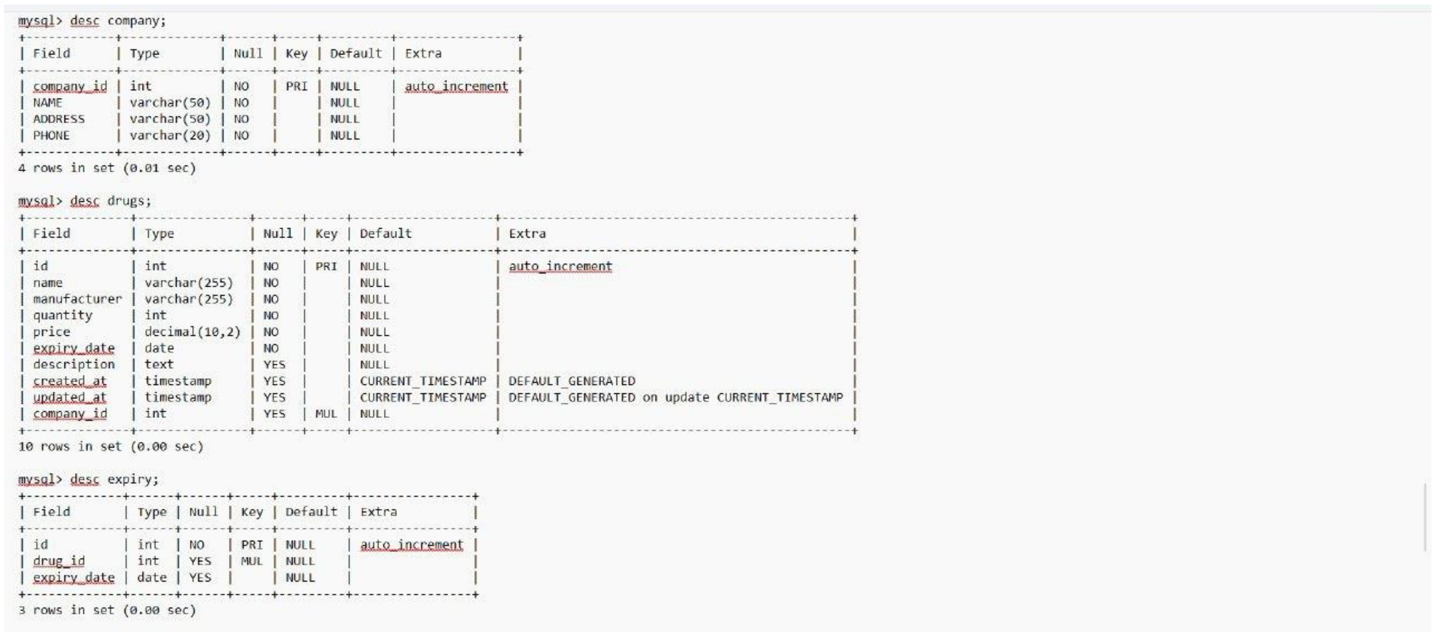
These tables provide information about history sales, inbox, and login within the system management framework. Beginning with the history sales table, attributes such as SaleID, Date, Time, and Total Price document past sales transactions. For inbox management, fields like MessageID, Sender, Recipient, Subject, and Message content can be included to track communication within the system. Meanwhile, the login table captures user authentication details, including fields like UserID, Username, Password, and Role. Connections between these tables can be established through foreign keys or additional relational tables, enabling functionalities such as associating sales transactions with specific users, tracking message exchanges between users, and managing user access through login credentials. This structured approach facilitates effective system management, ensuring accurate tracking of sales history, streamlined communication, and secure user authentication within the system.



```
 select * from company;
+------------+------------------+--------------------+------------------+
| company_id | NAME             | ADDRESS            | PHONE            |
+------------+------------------+--------------------+------------------+
|          1 | New Company Name | New Company Address | New Company Phone |
|          2 | XYZ Pharma       | 456 Elm Avenue     | +1987654321      |
|          3 | Generic Labs     | 789 Oak Lane       | +1122334455      |
|          4 | New Pharma       | Mumbai             | +9876543210      |
+------------+------------------+--------------------+------------------+
4 rows in set (0.02 sec)

mysql> -- Create a view for the company table
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW company_view AS
    -> SELECT *
    -> FROM company;
ERROR 1050 (42S01): Table 'company_view' already exists
mysql>
mysql> -- Display the contents of the company view
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM company_view;
+------------------+---------------------+-------------------+
| name             | address             | phone             |
+------------------+---------------------+-------------------+
| New Company Name | New Company Address | New Company Phone |
| XYZ Pharma       | 456 Elm Avenue      | +1987654321       |
| Generic Labs     | 789 Oak Lane        | +1122334455       |
| New Pharma       | Mumbai              | +9876543210       |
+------------------+---------------------+-------------------+
4 rows in set (0.00 sec)
```

Fig. 4.4

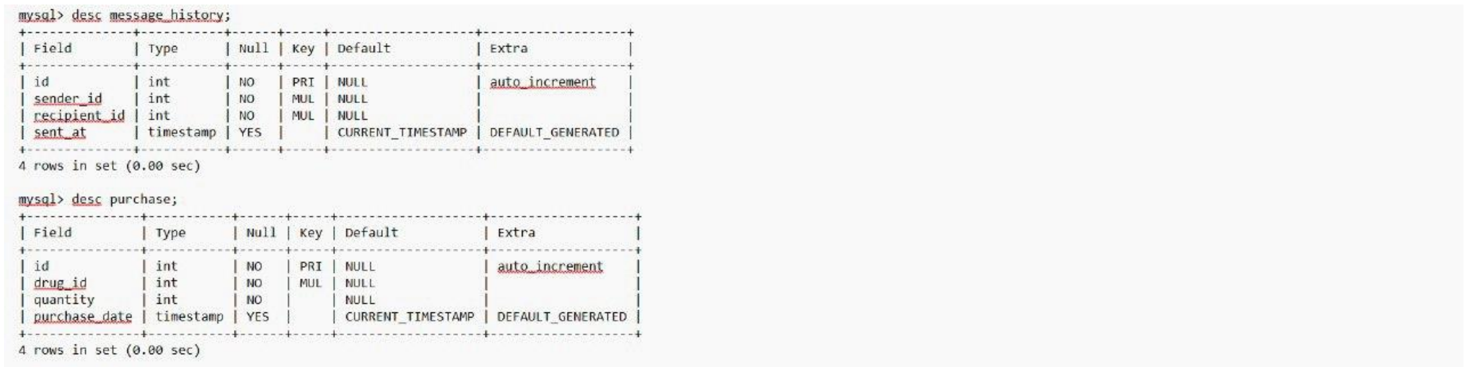
These tables provide information about companies and their company views for detailed tracking. Initially, the company table is established with attributes such as CompanyID, Name, Address, and Contact Information, storing data about various pharmaceutical manufacturers. Additionally, a company view can be created to offer a comprehensive overview, incorporating relevant fields from the company table along with aggregated data or additional information for enhanced analysis. Connections between these tables can be established through foreign keys or relational links, facilitating functionalities such as associating drugs with their respective manufacturers or tracking sales transactions associated with specific companies. This structured approach enables efficient tracking and analysis of company-related data within the system, empowering users to make informed decisions and effectively manage relationships with pharmaceutical suppliers.



```
mysql> desc company;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| company_id | int         | NO   | PRI | NULL    | auto_increment |
| NAME       | varchar(50) | NO   |     | NULL    |                |
| ADDRESS    | varchar(50) | NO   |     | NULL    |                |
| PHONE      | varchar(20) | NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> desc drugs;
+--------------+---------------+------+-----+-------------------+-----------------------------------------------+
| Field        | Type          | Null | Key | Default           | Extra                                         |
+--------------+---------------+------+-----+-------------------+-----------------------------------------------+
| id           | int           | NO   | PRI | NULL              | auto_increment                                |
| name         | varchar(255)  | NO   |     | NULL              |                                               |
| manufacturer | varchar(255)  | NO   |     | NULL              |                                               |
| quantity     | int           | NO   |     | NULL              |                                               |
| price        | decimal(10,2) | NO   |     | NULL              |                                               |
| expiry_date  | date          | NO   |     | NULL              |                                               |
| description  | text          | YES  |     | NULL              |                                               |
| created_at   | timestamp     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED                             |
| updated_at   | timestamp     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTAMP |
| company_id   | int           | YES  | MUL | NULL              |                                               |
+--------------+---------------+------+-----+-------------------+-----------------------------------------------+
10 rows in set (0.00 sec)

mysql> desc expiry;
+-------------+------+------+-----+---------+----------------+
| Field       | Type | Null | Key | Default | Extra          |
+-------------+------+------+-----+---------+----------------+
| id          | int  | NO   | PRI | NULL    | auto_increment |
| drug_id     | int  | YES  | MUL | NULL    |                |
| expiry_date | date | YES  |     | NULL    |                |
+-------------+------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

Fig. 4.5

These tables contain details of drugs, including their expiry dates, with careful monitoring by the company. Initially, a table for drugs is created, encompassing attributes such as DrugID, Name, Expiry Date, Price, and Quantity in Stock. Alongside, an additional field for Manufacturer or Company can denote the pharmaceutical company responsible for producing each drug. This enables efficient tracking of drugs and their respective expiry dates, crucial for maintaining inventory integrity and ensuring product efficacy. Through relational links or foreign keys, connections can be established to link each drug entry with its corresponding manufacturer, facilitating comprehensive monitoring and oversight. This structured approach supports diligent monitoring of drug expiry dates by the respective companies, promoting effective inventory management and product quality assurance within the system.



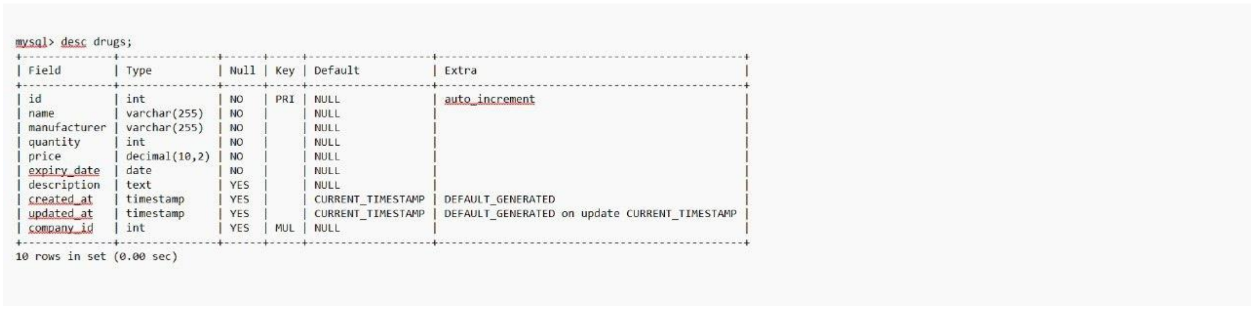
```
mysql> desc message_history;
+--------------+-----------+------+-----+-------------------+-------------------+
| Field        | Type      | Null | Key | Default           | Extra             |
+--------------+-----------+------+-----+-------------------+-------------------+
| id           | int       | NO   | PRI | NULL              | auto_increment    |
| sender_id    | int       | NO   | MUL | NULL              |                   |
| recipient_id | int       | NO   | MUL | NULL              |                   |
| sent_at      | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+--------------+-----------+------+-----+-------------------+-------------------+
4 rows in set (0.00 sec)

mysql> desc purchase;
+---------------+-----------+------+-----+-------------------+-------------------+
| Field         | Type      | Null | Key | Default           | Extra             |
+---------------+-----------+------+-----+-------------------+-------------------+
| id            | int       | NO   | PRI | NULL              | auto_increment    |
| drug_id       | int       | NO   | MUL | NULL              |                   |
| quantity      | int       | NO   |     | NULL              |                   |
| purchase_date | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+---------------+-----------+------+-----+-------------------+-------------------+
4 rows in set (0.00 sec)
```

Fig. 4.6

These tables store message history and purchase data within the database. Initially, a table for message history is established, containing attributes such as MessageID, Sender, Recipient, Subject, and Message Content, facilitating the tracking of communication within the system. Additionally, a separate table for purchase data is created, incorporating fields such as PurchaseID, ProductID, CustomerID, Quantity, and Total Price to record details of each transaction. Through relational links or foreign keys, connections can be established between these tables, enabling functionalities such as associating messages with specific purchases or tracking purchases made by individual customers. This structured approach ensures efficient storage and retrieval of message history and purchase data, supporting seamless communication and transaction management within the system.

5 Complex Queries Using SQL

5.1 Constraints

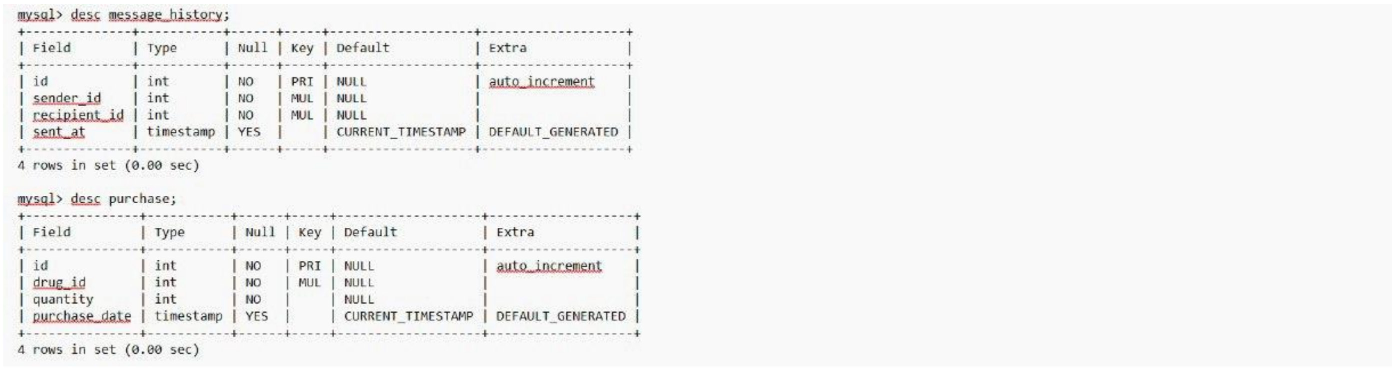


```
mysql> desc drugs;
+--------------+---------------+------+-----+-------------------+-----------------------------------------------+
| Field        | Type          | Null | Key | Default           | Extra                                         |
+--------------+---------------+------+-----+-------------------+-----------------------------------------------+
| id           | int           | NO   | PRI | NULL              | auto_increment                                |
| name         | varchar(255)  | NO   |     | NULL              |                                               |
| manufacturer | varchar(255)  | NO   |     | NULL              |                                               |
| quantity     | int           | NO   |     | NULL              |                                               |
| price        | decimal(10,2) | NO   |     | NULL              |                                               |
| expiry_date  | date          | NO   |     | NULL              |                                               |
| description  | text          | YES  |     | NULL              |                                               |
| created_at   | timestamp     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED                             |
| updated_at   | timestamp     | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTAMP |
| company_id   | int           | YES  | MUL | NULL              |                                               |
+--------------+---------------+------+-----+-------------------+-----------------------------------------------+
10 rows in set (0.00 sec)
```

Fig 5.1

In complex queries using SQL, constraints play a crucial role in ensuring data integrity and enforcing business rules within the database. These constraints define rules that data must adhere to, preventing inconsistencies or errors. For instance, in a pharmacy management system, constraints can be applied to tables such as Drugs, Sales, and Users to maintain accurate records. Common constraints include primary keys to enforce uniqueness, foreign keys to establish relationships between tables, and check constraints to enforce specific conditions on data values. Additionally, constraints such as NOT NULL ensure that essential fields are always populated, while UNIQUE constraints prevent duplicate entries. Through the implementation of constraints, SQL queries can confidently retrieve and manipulate data while adhering to predefined rules, ensuring the reliability and accuracy of information stored in the database

5.2 Triggers

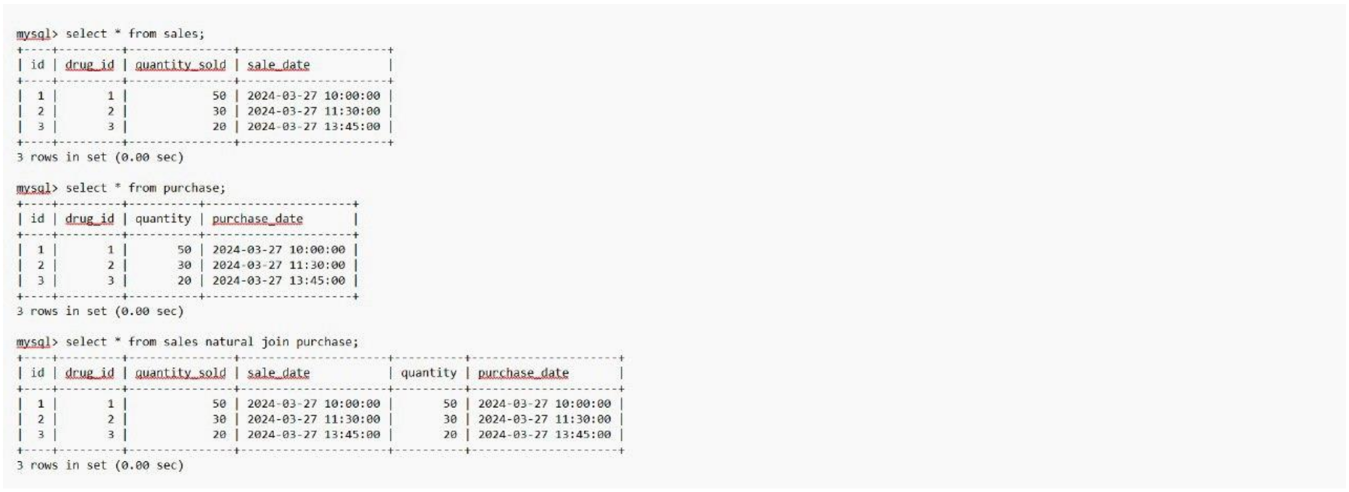


```
mysql> desc message_history;
+--------------+-----------+------+-----+-------------------+-------------------+
| Field        | Type      | Null | Key | Default           | Extra             |
+--------------+-----------+------+-----+-------------------+-------------------+
| id           | int       | NO   | PRI | NULL              | auto_increment    |
| sender_id    | int       | NO   | MUL | NULL              |                   |
| recipient_id | int       | NO   | MUL | NULL              |                   |
| sent_at      | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+--------------+-----------+------+-----+-------------------+-------------------+
4 rows in set (0.00 sec)

mysql> desc purchase;
+---------------+-----------+------+-----+-------------------+-------------------+
| Field         | Type      | Null | Key | Default           | Extra             |
+---------------+-----------+------+-----+-------------------+-------------------+
| id            | int       | NO   | PRI | NULL              | auto_increment    |
| drug_id       | int       | NO   | MUL | NULL              |                   |
| quantity      | int       | NO   |     | NULL              |                   |
| purchase_date | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+---------------+-----------+------+-----+-------------------+-------------------+
4 rows in set (0.00 sec)
```

Fig 5.2

In complex queries using SQL, triggers offer a powerful mechanism for automating actions in response to specific database events. These triggers can be defined to execute SQL statements before or after INSERT, UPDATE, or DELETE operations on tables. In the context of a pharmacy management system, triggers can be utilized to enforce complex business logic or perform additional tasks automatically. For example, a trigger can be implemented to update the stock quantity of drugs after a sales transaction occurs, ensuring accurate inventory management. Additionally, triggers can enforce data validation rules, such as verifying that certain conditions are met before allowing a transaction to proceed. By leveraging triggers, SQL queries gain enhanced functionality, enabling the system to react dynamically to changes in the database and ensuring consistent and reliable data processing.

5.3 Joins

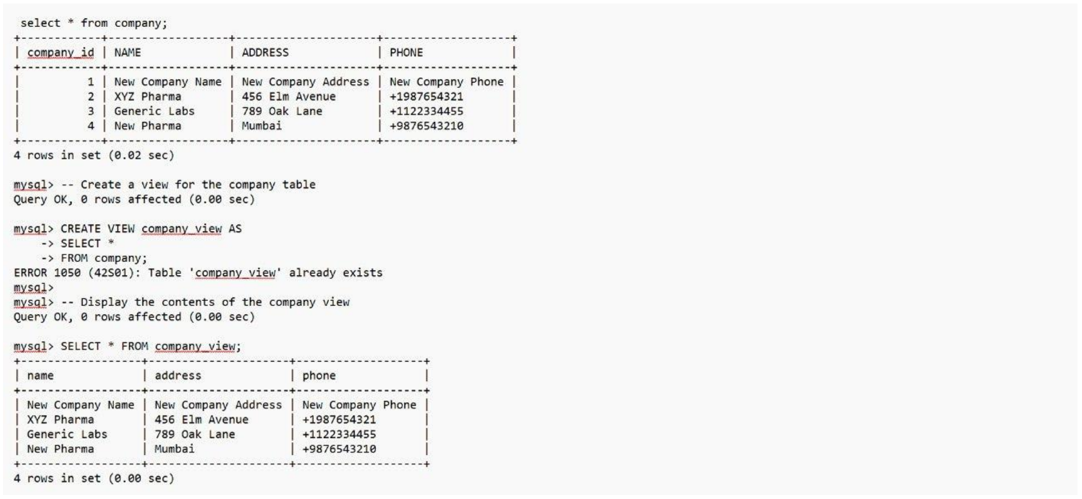


```
mysql> select * from sales;
+----+---------+---------------+---------------------+
| id | drug_id | quantity_sold | sale_date           |
+----+---------+---------------+---------------------+
|  1 |       1 |            50 | 2024-03-27 10:00:00 |
|  2 |       2 |            30 | 2024-03-27 11:30:00 |
|  3 |       3 |            20 | 2024-03-27 13:45:00 |
+----+---------+---------------+---------------------+
3 rows in set (0.00 sec)

mysql> select * from purchase;
+----+---------+----------+---------------------+
| id | drug_id | quantity | purchase_date       |
+----+---------+----------+---------------------+
|  1 |       1 |       50 | 2024-03-27 10:00:00 |
|  2 |       2 |       30 | 2024-03-27 11:30:00 |
|  3 |       3 |       20 | 2024-03-27 13:45:00 |
+----+---------+----------+---------------------+
3 rows in set (0.00 sec)

mysql> select * from sales natural join purchase;
+----+---------+---------------+---------------------+----------+---------------------+
| id | drug_id | quantity_sold | sale_date           | quantity | purchase_date       |
+----+---------+---------------+---------------------+----------+---------------------+
|  1 |       1 |            50 | 2024-03-27 10:00:00 |       50 | 2024-03-27 10:00:00 |
|  2 |       2 |            30 | 2024-03-27 11:30:00 |       30 | 2024-03-27 11:30:00 |
|  3 |       3 |            20 | 2024-03-27 13:45:00 |       20 | 2024-03-27 13:45:00 |
+----+---------+---------------+---------------------+----------+---------------------+
3 rows in set (0.00 sec)
```

Fig 5.3

In complex queries using SQL, joins are fundamental for retrieving data from multiple tables based on related columns. Joins enable the system to combine rows from two or more tables based on a related column between them. In a pharmacy management system, joins can be utilized to gather comprehensive information, such as linking sales transactions to specific customers or associating drugs with their respective manufacturers. Common types of joins include INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN, each offering different ways to combine data from multiple tables. For instance, an INNER JOIN returns rows where there is a match in both tables, while a LEFT JOIN retrieves all rows from the left table and matching rows from the right table. By leveraging joins effectively in SQL queries, the system can gather interconnected data seamlessly, providing valuable insights and facilitating informed decision-making processes.

5.4 Views

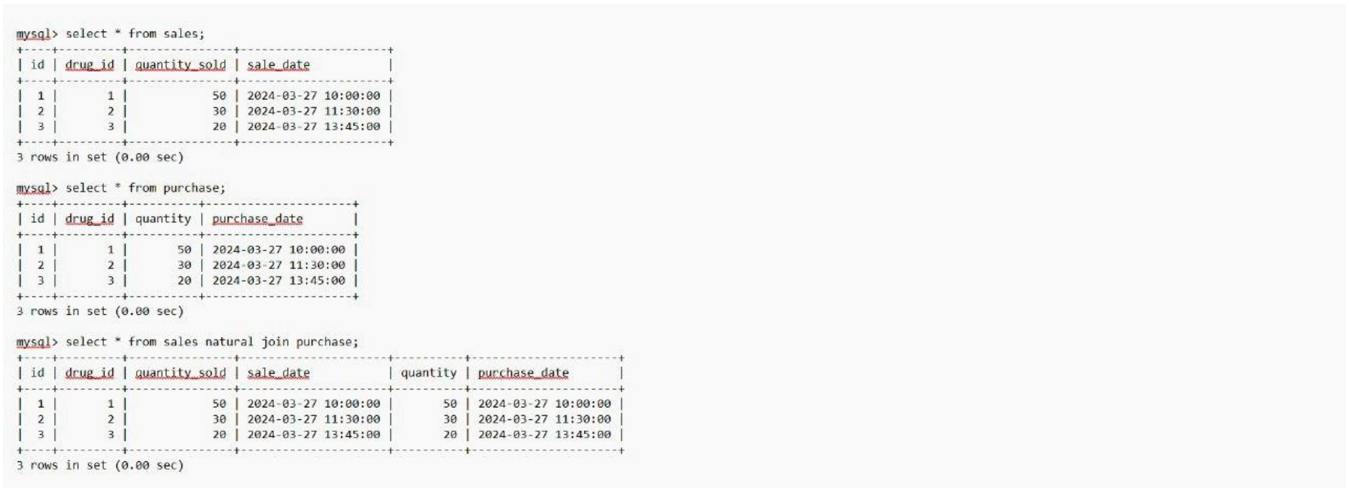


```
 select * from company;
+------------+------------------+---------------------+-------------------+
| company_id | NAME             | ADDRESS             | PHONE             |
+------------+------------------+---------------------+-------------------+
|          1 | New Company Name | New Company Address | New Company Phone |
|          2 | XYZ Pharma       | 456 Elm Avenue      | +1987654321       |
|          3 | Generic Labs     | 789 Oak Lane        | +1122334455       |
|          4 | New Pharma       | Mumbai              | +9876543210       |
+------------+------------------+---------------------+-------------------+
4 rows in set (0.02 sec)

mysql> -- Create a view for the company table
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW company_view AS
    -> SELECT *
    -> FROM company;
ERROR 1050 (42S01): Table 'company_view' already exists
mysql>
mysql> -- Display the contents of the company view
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM company_view;
+------------------+---------------------+-------------------+
| name             | address             | phone             |
+------------------+---------------------+-------------------+
| New Company Name | New Company Address | New Company Phone |
| XYZ Pharma       | 456 Elm Avenue      | +1987654321       |
| Generic Labs     | 789 Oak Lane        | +1122334455       |
| New Pharma       | Mumbai              | +9876543210       |
+------------------+---------------------+-------------------+
4 rows in set (0.00 sec)
```

Fig 5.4

In complex queries using SQL, views provide a valuable mechanism for simplifying data access and enhancing query efficiency. Views are virtual tables that represent the result of a predefined SQL query, allowing users to interact with complex data structures as if they were simple tables. In a pharmacy management system, views can be created to present consolidated information, such as a summary of sales transactions or a list of available drugs with relevant details. Views abstract away the underlying complexity of data relationships and transformations, making it easier for users to retrieve the information they need without having to write intricate queries each time. Additionally, views can be used to enforce security measures by limiting access to sensitive data or presenting a subset of data tailored to specific user roles. By leveraging views effectively, SQL queries can become more concise, maintainable, and user-friendly, ultimately enhancing the overall usability and performance of the system.

5.5 Store Procedure

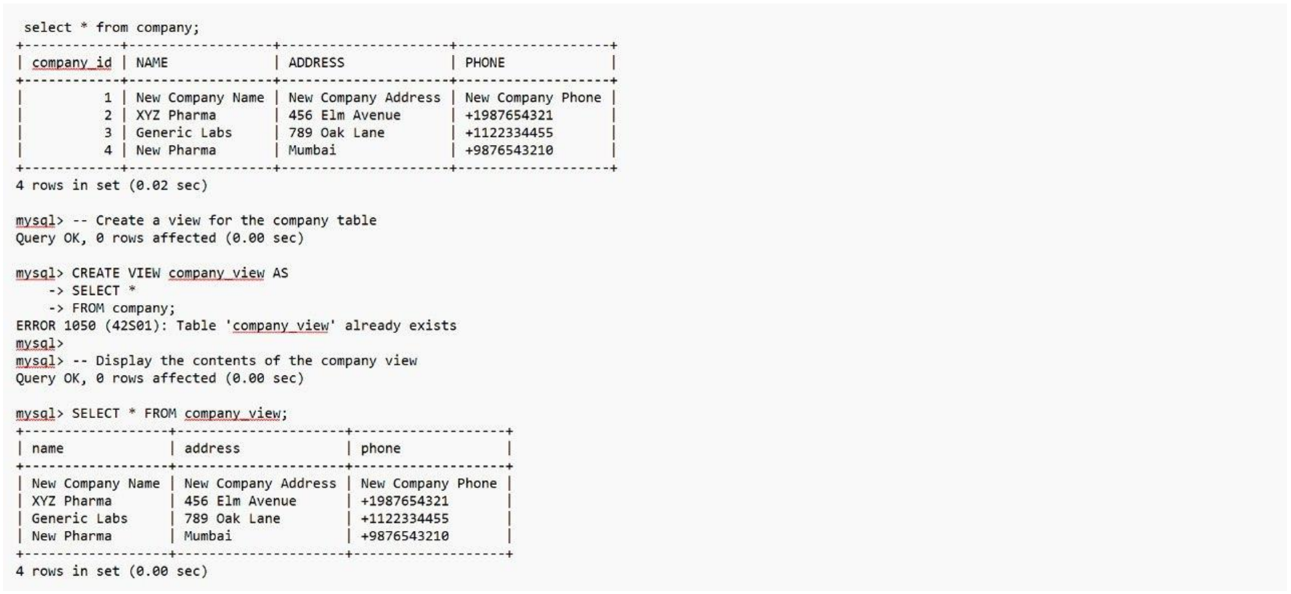


```
mysql> select * from sales;
+----+---------+---------------+---------------------+
| id | drug_id | quantity_sold | sale_date           |
+----+---------+---------------+---------------------+
|  1 |       1 |            50 | 2024-03-27 10:00:00 |
|  2 |       2 |            30 | 2024-03-27 11:30:00 |
|  3 |       3 |            20 | 2024-03-27 13:45:00 |
+----+---------+---------------+---------------------+
3 rows in set (0.00 sec)

mysql> select * from purchase;
+----+---------+----------+---------------------+
| id | drug_id | quantity | purchase_date       |
+----+---------+----------+---------------------+
|  1 |       1 |       50 | 2024-03-27 10:00:00 |
|  2 |       2 |       30 | 2024-03-27 11:30:00 |
|  3 |       3 |       20 | 2024-03-27 13:45:00 |
+----+---------+----------+---------------------+
3 rows in set (0.00 sec)

mysql> select * from sales natural join purchase;
+----+---------+---------------+---------------------+----------+---------------------+
| id | drug_id | quantity_sold | sale_date           | quantity | purchase_date       |
+----+---------+---------------+---------------------+----------+---------------------+
|  1 |       1 |            50 | 2024-03-27 10:00:00 |       50 | 2024-03-27 10:00:00 |
|  2 |       2 |            30 | 2024-03-27 11:30:00 |       30 | 2024-03-27 11:30:00 |
|  3 |       3 |            20 | 2024-03-27 13:45:00 |       20 | 2024-03-27 13:45:00 |
+----+---------+---------------+---------------------+----------+---------------------+
3 rows in set (0.00 sec)
```

Fig 5.5

In complex queries using SQL, stored procedures offer a powerful tool for encapsulating and executing sets of SQL statements. Stored procedures are precompiled SQL code blocks that can be stored and executed on the database server. In a pharmacy management system, stored procedures can be utilized to perform common tasks, such as adding new drugs to the inventory, processing sales transactions, or generating reports. By encapsulating these tasks into stored procedures, users can execute complex operations with a single command, reducing the need to write repetitive SQL queries. Stored procedures also enhance security by controlling access to sensitive database operations and enforcing business logic at the database level. Additionally, stored procedures can improve performance by reducing network traffic and optimizing query execution plans. Overall, stored procedures streamline database interactions, enhance security, and improve performance in complex SQL queries within the pharmacy management system.

5.6 Cursor



```
 select * from company;
+------------+------------------+---------------------+-------------------+
| company_id | NAME             | ADDRESS             | PHONE             |
+------------+------------------+---------------------+-------------------+
|          1 | New Company Name | New Company Address | New Company Phone |
|          2 | XYZ Pharma       | 456 Elm Avenue      | +1987654321       |
|          3 | Generic Labs     | 789 Oak Lane        | +1122334455       |
|          4 | New Pharma       | Mumbai              | +9876543210       |
+------------+------------------+---------------------+-------------------+
4 rows in set (0.02 sec)

mysql> -- Create a view for the company table
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW company_view AS
    -> SELECT *
    -> FROM company;
ERROR 1050 (42S01): Table 'company_view' already exists
mysql>
mysql> -- Display the contents of the company view
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM company_view;
+------------------+---------------------+-------------------+
| name             | address             | phone             |
+------------------+---------------------+-------------------+
| New Company Name | New Company Address | New Company Phone |
| XYZ Pharma       | 456 Elm Avenue      | +1987654321       |
| Generic Labs     | 789 Oak Lane        | +1122334455       |
| New Pharma       | Mumbai              | +9876543210       |
+------------------+---------------------+-------------------+
4 rows in set (0.00 sec)
```

Fig 5.6

In complex queries using SQL, cursors provide a mechanism for iterating over a result set row by row. Cursors are particularly useful when dealing with complex data processing tasks that require sequential access to individual rows of data. In a pharmacy management system, cursors can be employed to perform operations such as calculating aggregate statistics, updating records based on specific criteria, or iterating through sales transactions for analysis. By using cursors, developers can control the flow of data processing and apply custom logic to each row as needed. However, it's important to note that cursors can have performance implications, as they may involve additional overhead compared to set-based operations. Therefore, they should be used judiciously, particularly in scenarios where set-based operations are not feasible or efficient. Overall, cursors offer a flexible and powerful tool for implementing complex data processing tasks within SQL queries, enhancing the functionality and versatility of the pharmacy management system.

6 Pitfalls and Normalizations

6.1 Analyzing the Pitfalls

Redundancy

Redundancy in database design leads to data inconsistencies, increased storage requirements, and risks of anomalies during updates, insertions, and deletions. It complicates data maintenance and management, requiring extra effort to ensure consistency across redundant copies. To mitigate these issues, database designers should aim for proper normalization to minimize redundancy and dependency, and enforce referential integrity constraints to maintain data consistency.

Inconsistency

Inconsistency in database design arises from redundant data, leading to discrepancies between copies. It results in challenges in data management, with updates made to one copy not reflecting in others, risking data integrity. Addressing redundancy through normalization and enforcing constraints helps mitigate inconsistency issues, ensuring data accuracy and reliability.

Inefficiency

Inefficiency in database design stems from poor indexing, over-normalization, and inadequate query optimization. It leads to slow query performance, resource wastage, and scalability limitations. Prioritizing indexing on frequently queried columns, balancing normalization with performance, and optimizing queries can mitigate inefficiency, enhancing overall database performance.

Complexity

Complexity in database design results from over-normalization, inconsistent naming conventions, and inadequate documentation. It leads to difficulties in understanding and maintaining the database schema, increasing the likelihood of errors and inefficiencies. Simplifying normalization, establishing clear naming conventions, and thorough documentation can alleviate complexity, facilitating easier database management and development.

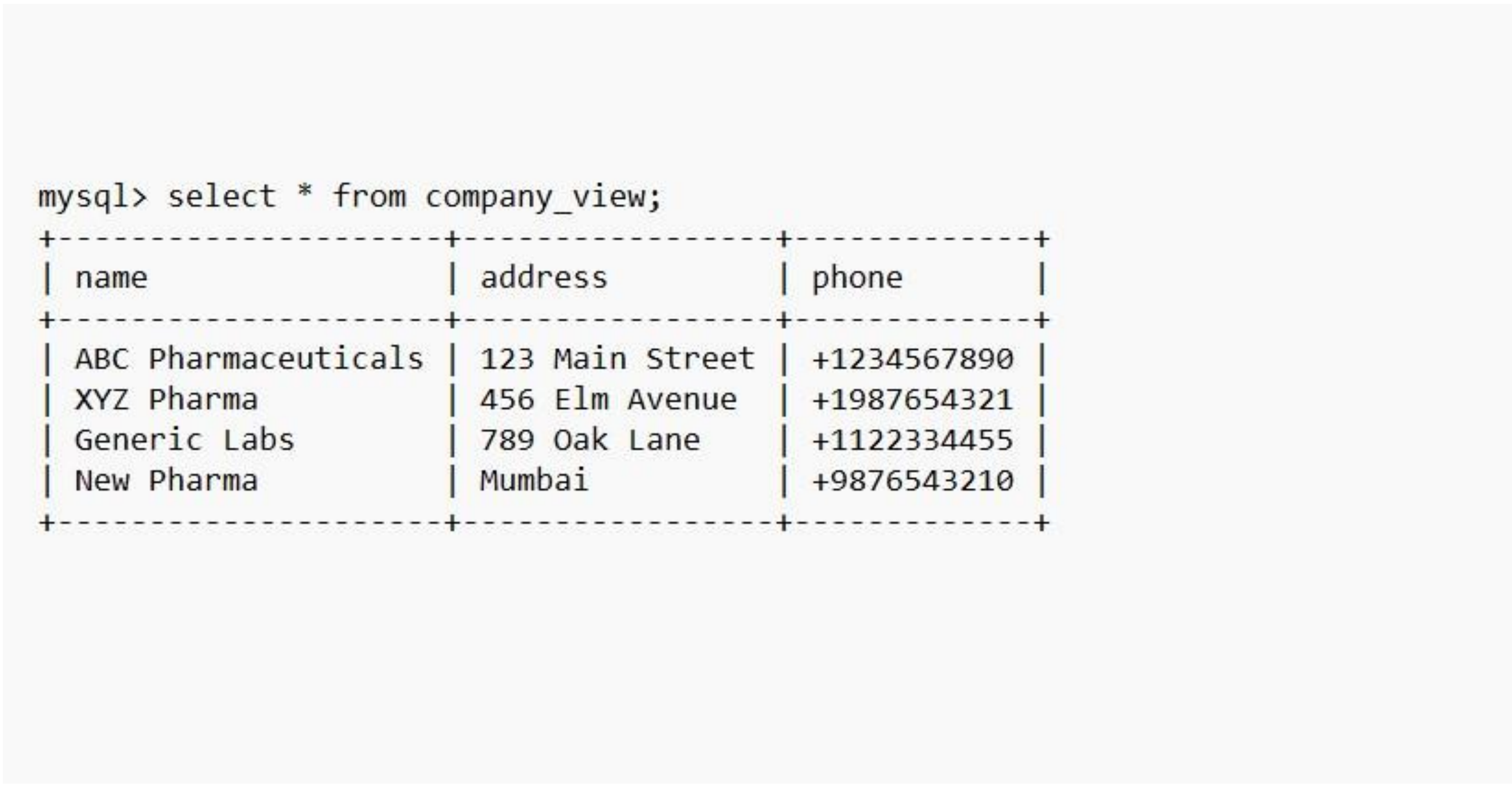
6.2 Identifying the Dependencies

Functional dependency in a database means that if you know the value of one attribute, you can predict the value of another attribute. It's like a rule that tells us how one piece of information relates to another in a table. This concept helps organize data efficiently and avoid repeating the same information unnecessarily.

Functional dependency (FD) is a constraint that specifies the relationship between two attributes in a database table.

Functional Dependency:

+



```
mysql> select * from company_view;
+--------------------+----------------+------------+
| name               | address        | phone      |
+--------------------+----------------+------------+
| ABC Pharmaceuticals | 123 Main Street | +1234567890 |
| XYZ Pharma         | 456 Elm Avenue | +1987654321 |
| Generic Labs       | 789 Oak Lane   | +1122334455 |
| New Pharma         | Mumbai         | +9876543210 |
+--------------------+----------------+------------+
```

Fig 6.1

Functional Dependency:

COMPANY_ID → NAME, ADDRESS, PHONE

The COMPANY_ID uniquely identifies each company, and it determines the values of NAME, ADDRESS, and PHONE.

NAME → COMPANY_ID, ADDRESS, PHONE

The NAME of the company also uniquely identifies each company, and it determines the values of COMPANY_ID, ADDRESS, and PHONE.

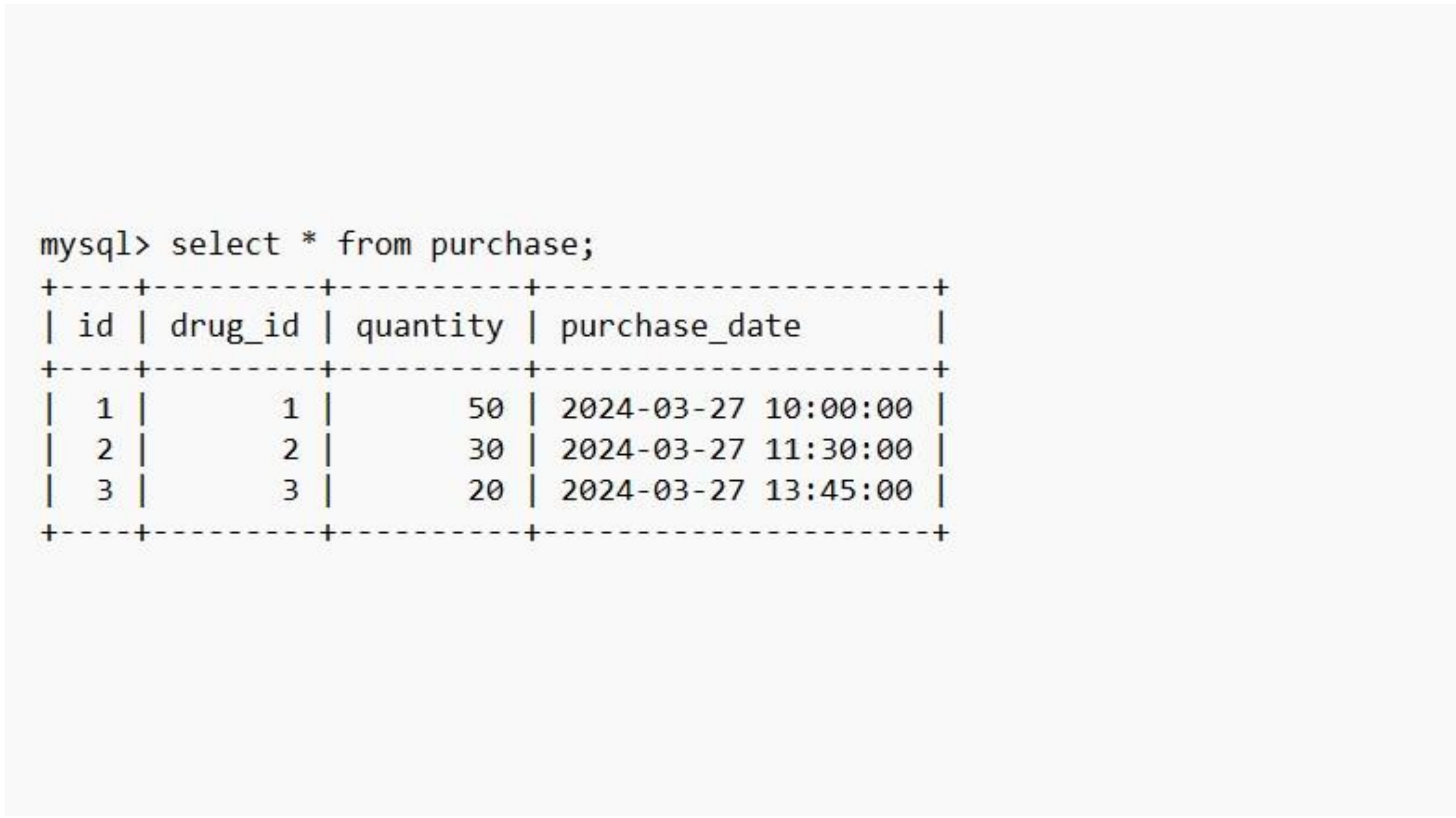
ADDRESS → COMPANY_ID, NAME, PHONE

Each address likely corresponds to a single company, so ADDRESS determines the values of COMPANY_ID, NAME, and PHONE.

PHONE → COMPANY_ID, NAME, ADDRESS

Similarly, each phone number is likely associated with a single company, so PHONE determines the values of COMPANY_ID, NAME, and ADDRESS.

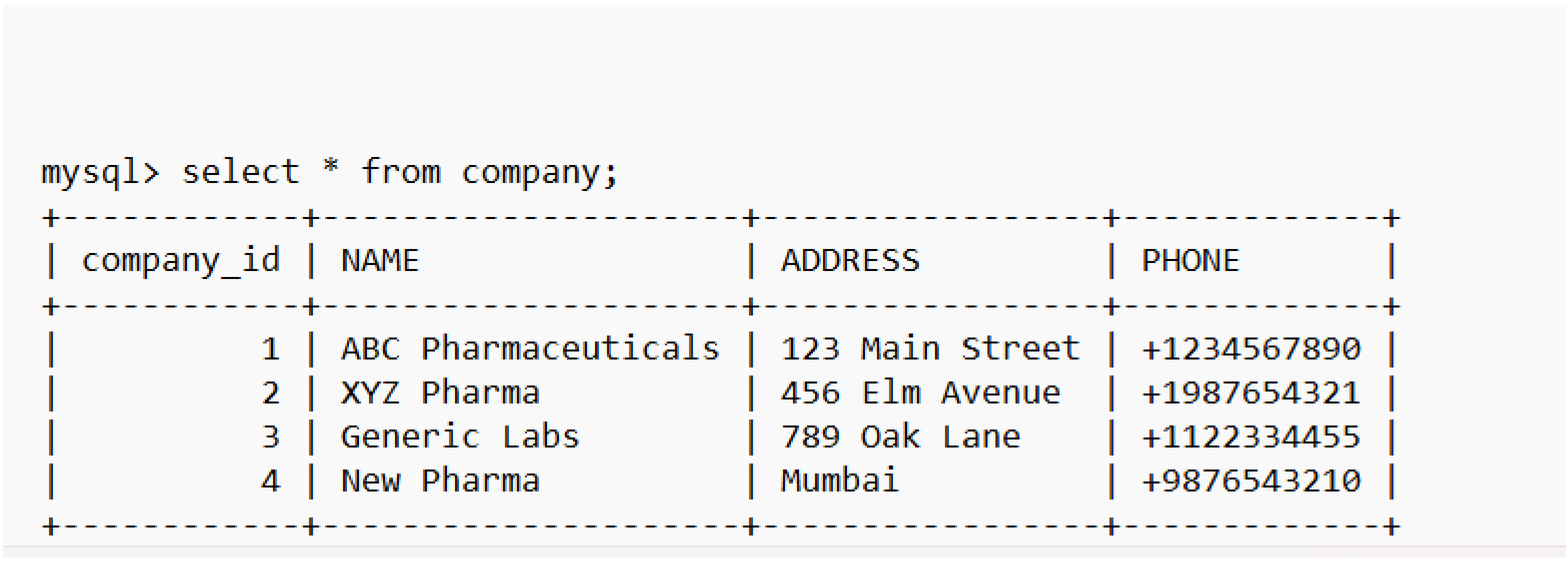
Generated Functional Dependencies:
COMPANY_ID → NAME
COMPANY_ID → ADDRESS
COMPANY_ID → PHONE
NAME → COMPANY_ID
NAME → ADDRESS
NAME → PHONE
ADDRESS → COMPANY_ID
ADDRESS → NAME
ADDRESS → PHONE
PHONE → COMPANY_ID
PHONE → NAME
PHONE → ADDRESS



```
mysql> select * from purchase;
+----+---------+----------+---------------------+
| id | drug_id | quantity | purchase_date       |
+----+---------+----------+---------------------+
|  1 |       1 |       50 | 2024-03-27 10:00:00 |
|  2 |       2 |       30 | 2024-03-27 11:30:00 |
|  3 |       3 |       20 | 2024-03-27 13:45:00 |
+----+---------+----------+---------------------+
```

Fig 6.2

Functional Dependency:
id → drug_id
Each purchase ID uniquely determines the corresponding drug ID.
id → quantity
Each purchase ID uniquely determines the quantity of drugs purchased.
id → purchase_date
Each purchase ID uniquely determines the purchase date.
Generated Functional Dependencies:
id → drug_id
id → quantity
id → purchase_date



```
mysql> select * from company;
+------------+----------------------+-----------------+--------------+
| company_id | NAME                 | ADDRESS         | PHONE        |
+------------+----------------------+-----------------+--------------+
|          1 | ABC Pharmaceuticals  | 123 Main Street | +1234567890  |
|          2 | XYZ Pharma           | 456 Elm Avenue  | +1987654321  |
|          3 | Generic Labs         | 789 Oak Lane    | +1122334455  |
|          4 | New Pharma           | Mumbai          | +9876543210  |
+------------+----------------------+-----------------+--------------+
```

Fig 6.3

Functional Dependency:
COMPANY_ID → NAME
Each company ID uniquely determines the corresponding company name.
COMPANY_ID → ADDRESS
Each company ID uniquely determines the corresponding company address.
COMPANY_ID → PHONE
Each company ID uniquely determines the corresponding company phone number.
NAME → COMPANY_ID
Each company name uniquely determines the corresponding company ID.
ADDRESS → COMPANY_ID
Each company address uniquely determines the corresponding company ID.
PHONE → COMPANY_ID
Each company phone number uniquely determines the corresponding company ID.
Generated Functional Dependencies:
COMPANY_ID → NAME

COMPANY_ID → ADDRESS
COMPANY_ID → PHONE
NAME → COMPANY_ID
ADDRESS → COMPANY_ID
PHONE → COMPANY_ID

6.3 Applying Normalizations

Normalization in databases is the process of organizing data in a relational database to reduce redundancy and dependency. It involves breaking down large tables into smaller, more manageable tables and defining relationships between them. Normalization typically involves several normal forms (1NF, 2NF, 3NF, BCNF, etc.) to ensure data integrity and optimize database performance. The goal of normalization is to minimize data redundancy, improve data integrity, and make the database structure more flexible and adaptable to changes.

FIRST NORMAL FORM (1NF)

A relation is said to be in First Normal Form if and only if all the attributes of relation are atomic (Single valued) in nature. It must not contain any multi valued (or) composite attributes.

TABLES IN DATABASE:

1. Table: `users`

The table is in 1NF since each cell contains a single value.

2. Table: `drugs`

The table is in 1NF.

3. Table: `company view`

The table is in 1NF.

4. Table: `expiry `

The table is in 1NF.

5. Table: `history_sales`

The table is in 1NF.

6. Table: `inbox`

The table is in 1NF.

7. Table: `purchase`

The table is in 1NF.

SECOND NORMAL FORM (2NF)

A relation is said to be in Second Normal Form, If and only if it is already in First Normal Form. There exists no Partial Dependency.

<u>TABLES IN DATABASE:</u>

1. Table: `users`

It's also in 2NF because there are no partial dependencies; all non-prime attributes are fully functionally dependent on the primary key.

2. Table: `drugs`

It's in 2NF because there are no partial dependencies.

3. Table: `company_view`

It's in 2NF because there are no partial dependencies.

4. Table: `expiry`

It's in 2NF because there are no partial dependencies.

5. Table: `history_sales`

It's in 2NF because there are no partial dependencies.

6. Table: `inbox`

It's in 2NF because there are no partial dependencies.

7. Table: `purchase `

It's in 2NF because there are no partial dependencies.

THIRD NORMAL FORM (3NF)

A relation will be in 3NF if it is in 2NF and does not contain any transitive partial dependency.

<u>TABLES IN DATABASE:</u>

1. Table: `users`

This table is in 3NF as well because there are no transitive dependencies.

2. Table: `drugs`

This table is in 3NF as well.

3. Table: `company_view`

This table is in 3NF as well.

4. Table: `expiry`

This table is in 3NF as well.

5. Table: `history_sales`

This table is in 3NF as well.

6. Table: `inbox`

This table is in 3NF as well.

7. Table: `purchase `

This table is in 3NF as well.

BOYCE CODD NORMAL FORM (BCNF)

BCNF is the advanced version of 3NF. It is stricter than 3NF. A table is in BCNF if every functional dependency X → Y, X is the super key of the table.

TABLES IN DATABASE:

1. Table: `users`

It's already in BCNF because there are no non-trivial functional dependencies where the determinant is not a superkey.

2. Table: `drugs`

It's already in BCNF.

3. Table: `company_view`

It's already in BCNF.

4. Table: `expiry`

It's already in BCNF.

5. Table: `history_sales`

It's already in BCNF.

6. Table: `inbox`

It's already in BCNF.

7. Table: `purchase`

It's already in BCNF.

FOURTH NORMAL FORM (4NF)

A relation will be in 4NF if it is in Boyce Codd normal form and has no multivalued dependency. For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multivalued dependency.

TABLES IN DATABASE:

1. Table: `users`

Since there are no multivalued dependencies, it is automatically in 4NF.

2. Table: `drugs`

Since there are no multivalued dependencies, it is automatically in 4NF.

3. Table: `company_view`

Since there are no multivalued dependencies, it is automatically in 4NF.

4. Table: `expiry`

Since there are no multivalued dependencies, it is automatically in 4NF.

5. Table: `history_sales`

Since there are no multivalued dependencies, it is automatically in 4NF.

6. Table: `inbox`

Since there are no multivalued dependencies, it is automatically in 4NF.

7. Table: `purchase`

Since there are no multivalued dependencies, it is automatically in 4NF.

FIFTH NORMAL FORM (5NF)

A relation is in 5NF if it is in 4NF and does not contain any join dependency and joining should be lossless. 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.

<u>TABLES IN DATABASE:</u>

1. Table: `users`

Also, there are no join dependencies, so it's automatically in 5NF.

2. Table: `drugs`

No join dependencies are present, so it's automatically in 5NF.

3. Table: `company_view`

No join dependencies are present, so it's automatically in 5NF.

4. Table: `expiry`

No join dependencies are present, so it's automatically in 5NF.

5. Table: `history_sales`

No join dependencies are present, so it's automatically in 5NF.

6. Table: `inbox`

No join dependencies are present, so it's automatically in 5NF.

7. Table: `purchase`

No join dependencies are present, so it's automatically in 5NF.

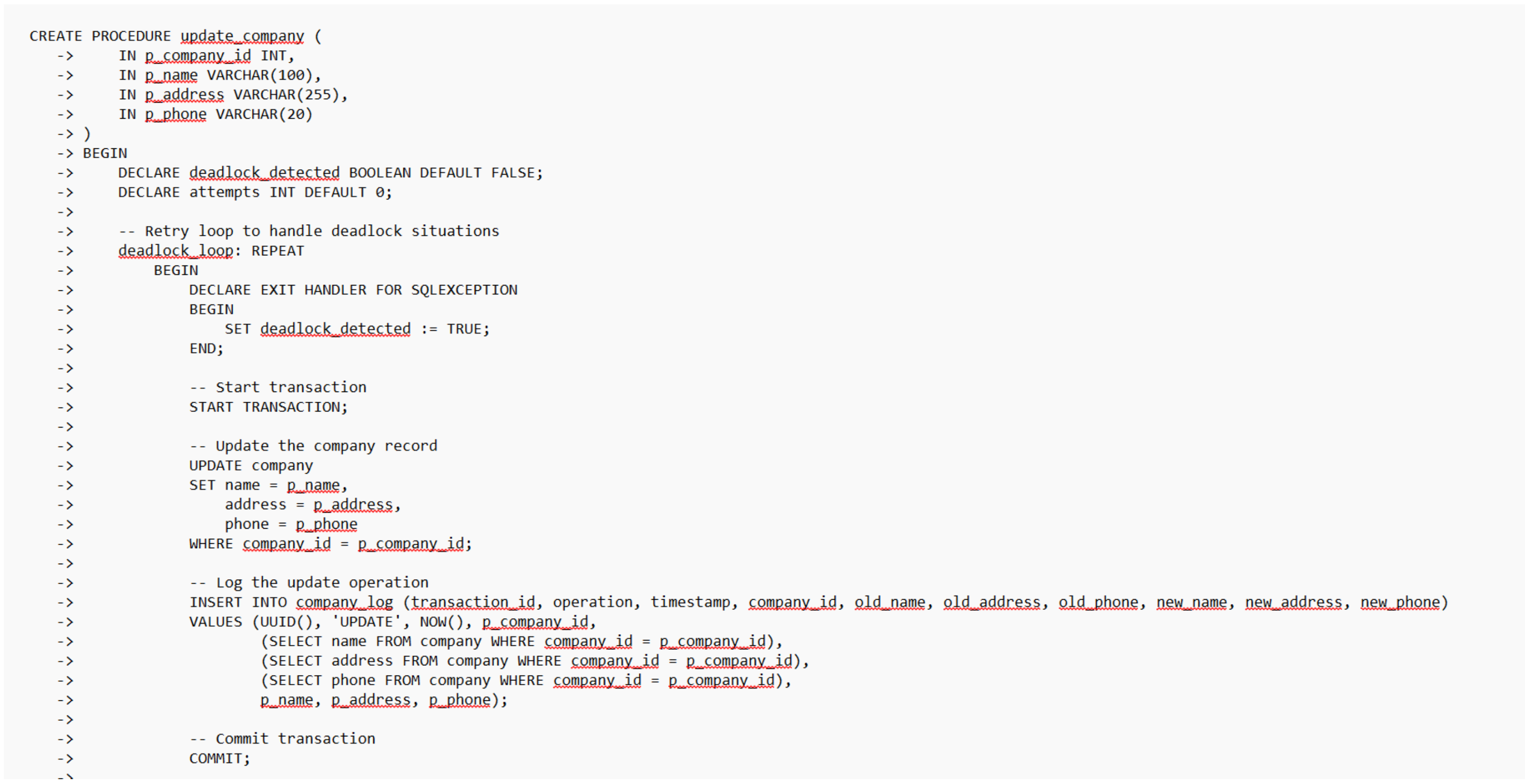
7 Implementation of Concurrency Control and Recovery Mechanisms

Concurrency Control

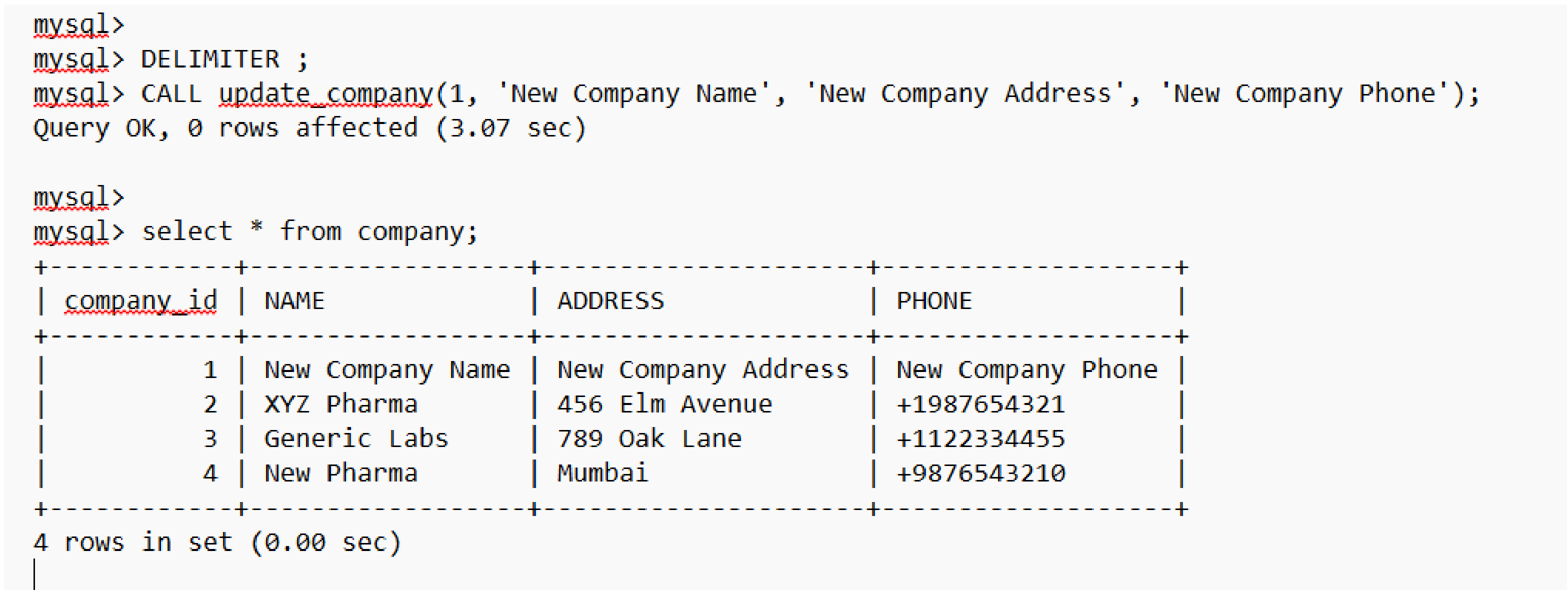
- Concurrency control in DBMS ensures multiple transactions execute together while maintaining data consistency. Techniques include:
- Locking: Transactions acquire locks on data items to prevent concurrent access.
- Two-Phase Locking (2PL): Prevents deadlocks by acquiring and releasing locks in two phases.
- Timestamp Ordering: Orders transactions based on unique timestamps, resolving conflicts.
- Multiversion Concurrency Control (MVCC): Maintains consistent snapshots by storing multiple data versions.
- Optimistic Concurrency Control: Allows transactions to proceed without initial locks, resolving conflicts at the end.
- Serializability: Ensures equivalent outcomes of concurrent transactions to serial execution. Choice of technique depends on application requirements and system constraints.

Recovery Mechanisms

- Recovery mechanisms in DBMS ensure data durability and consistency in the event of failures. Techniques include:
- Checkpoints: Periodic saving of the database state to reduce recovery time.
- Undo/Redo Logging: Records both before and after images of data modifications for recovery.
- Shadow Paging: Maintains a shadow copy of the database, updating it with transactions and swapping it with the original on commit.
- Deferred Update: Updates are made directly to memory, with changes logged and applied to the database on commit.
- Immediate Update: Updates are immediately made to the database, with undo logs kept for recovery. Choice of mechanism depends on factors like recovery time requirements and system.



```
CREATE PROCEDURE update_company (
    ->     IN p_company_id INT,
    ->     IN p_name VARCHAR(100),
    ->     IN p_address VARCHAR(255),
    ->     IN p_phone VARCHAR(20)
    -> )
    -> BEGIN
    ->     DECLARE deadlock_detected BOOLEAN DEFAULT FALSE;
    ->     DECLARE attempts INT DEFAULT 0;
    ->
    ->     -- Retry loop to handle deadlock situations
    ->     deadlock_loop: REPEAT
    ->         BEGIN
    ->             DECLARE EXIT HANDLER FOR SQLEXCEPTION
    ->             BEGIN
    ->                 SET deadlock_detected := TRUE;
    ->             END;
    ->
    ->             -- Start transaction
    ->             START TRANSACTION;
    ->
    ->             -- Update the company record
    ->             UPDATE company
    ->             SET name = p_name,
    ->                 address = p_address,
    ->                 phone = p_phone
    ->             WHERE company_id = p_company_id;
    ->
    ->             -- Log the update operation
    ->             INSERT INTO company_log (transaction_id, operation, timestamp, company_id, old_name, old_address, old_phone, new_name, new_address, new_phone)
    ->             VALUES (UUID(), 'UPDATE', NOW(), p_company_id,
    ->                     (SELECT name FROM company WHERE company_id = p_company_id),
    ->                     (SELECT address FROM company WHERE company_id = p_company_id),
    ->                     (SELECT phone FROM company WHERE company_id = p_company_id),
    ->                     p_name, p_address, p_phone);
    ->
    ->             -- Commit transaction
    ->             COMMIT;
    ->
```



```
mysql>
mysql> DELIMITER ;
mysql> CALL update_company(1, 'New Company Name', 'New Company Address', 'New Company Phone');
Query OK, 0 rows affected (3.07 sec)

mysql>
mysql> select * from company;
+------------+------------------+---------------------+-------------------+
| company_id | NAME             | ADDRESS             | PHONE             |
+------------+------------------+---------------------+-------------------+
|          1 | New Company Name | New Company Address | New Company Phone |
|          2 | XYZ Pharma       | 456 Elm Avenue      | +1987654321       |
|          3 | Generic Labs     | 789 Oak Lane        | +1122334455       |
|          4 | New Pharma       | Mumbai              | +9876543210       |
+------------+------------------+---------------------+-------------------+
4 rows in set (0.00 sec)
```

Fig 7.1

8 Code of the Project:

```
from flask import Flask, jsonify, redirect, render_template, request, url_for
import mysql.connector
import config

app = Flask(__name__)

db = mysql.connector.connect(**config.db_config)

ADMIN_USERNAME = "admin"
ADMIN_PASSWORD = "admin"


def get_sales_history():
    cursor = db.cursor(dictionary=True)
    cursor.execute("SELECT * FROM HISTORY_SALE")
    sales_history = cursor.fetchall()
    cursor.close()
    return sales_history


def get_purchases():
    cursor = db.cursor(dictionary=True)
    cursor.execute("SELECT * FROM PURCHASE")
    purchases = cursor.fetchall()
    cursor.close()
    return purchases


def get_companies():
    cursor = db.cursor(dictionary=True)
    cursor.execute("SELECT * FROM COMPANY")
    companies = cursor.fetchall()
    cursor.close()
    return companies


def get_drugs():
    cursor = db.cursor(dictionary=True)
    cursor.execute("SELECT * FROM DRUG")
    drugs = cursor.fetchall()
    cursor.close()
    return drugs


def get_sales_history_today():
    cursor = db.cursor(dictionary=True)
    cursor.execute("SELECT * FROM HISTORY_SALE WHERE DATE = CURDATE()")
    sales_history_today = cursor.fetchall()
    cursor.close()
    return sales_history_today


def get_sales_history_month():
    cursor = db.cursor(dictionary=True)
    cursor.execute(
        "SELECT * FROM HISTORY_SALE WHERE MONTH(DATE) =
MONTH(CURDATE()) AND YEAR(DATE) = YEAR(CURDATE())"
    )
    sales_history_month = cursor.fetchall()
    cursor.close()
    return sales_history_month


@app.route("/login")
def login():
    return render_template("login.html")


@app.route("/login", methods=["POST"])
def authenticate():
    username = request.form.get("username")
    password = request.form.get("password")

    if username == ADMIN_USERNAME and password == ADMIN_PASSWORD:
        return redirect(url_for("admin"))
    else:
        return render_template("login.html", error="Invalid username or password")


@app.route("/admin")
def admin():
    companies = get_companies()
    drugs = get_drugs()
    return render_template("admin.html", companies=companies, drugs=drugs)


@app.route("/add_company", methods=["POST"])
def add_company():
    if request.method == "POST":
        company_name = request.form.get("company_name")
        company_address = request.form.get("company_address")
        company_phone = request.form.get("company_phone")

        cursor = db.cursor()
        try:
            cursor.execute(
                """
                INSERT INTO COMPANY (NAME, ADDRESS, PHONE)
                VALUES (%s, %s, %s)
                """,
                (company_name, company_address, company_phone),
            )
            db.commit()
        except mysql.connector.Error as err:
            db.rollback()
            print("Failed to add company:", err)
        finally:
            cursor.close()

        return redirect(url_for("admin"))


@app.route("/edit_company", methods=["POST"])
def edit_company():
    if request.method == "POST":
        company_name = request.form.get("company_name")
        company_address = request.form.get("company_address")
        company_phone = request.form.get("company_phone")

        cursor = db.cursor()
        try:
            cursor.execute(
                """
                UPDATE COMPANY
                SET ADDRESS = %s, PHONE = %s
                WHERE NAME = %s
                """,
                (company_address, company_phone, company_name),
            )
            db.commit()
        except mysql.connector.Error as err:
            db.rollback()
            print("Failed to update company:", err)
        finally:
            cursor.close()

        return redirect(url_for("admin"))


@app.route("/delete_company/<company_name>", methods=["DELETE"])
def delete_company(company_name):
    cursor = db.cursor()
    try:
        cursor.execute("DELETE FROM COMPANY WHERE NAME = %s",
(company_name,))
        db.commit()
        return jsonify(success=True)
    except Exception as e:
        print("Error:", e)
        db.rollback()
        return jsonify(success=False, error=str(e)), 500
    finally:
        cursor.close()


@app.route("/add_drug", methods=["POST"])
def add_drug():
    if request.method == "POST":
        drug_name = request.form.get("drug_name")
        drug_type = request.form.get("drug_type")
        drug_barcode = request.form.get("drug_barcode")
        drug_quantity = request.form.get("drug_quantity")

        cursor = db.cursor()
        try:
            cursor.execute(
                """
                INSERT INTO DRUG (NAME, TYPE, BARCODE, QUANTITY)
                VALUES (%s, %s, %s, %s)
                """,
                (drug_name, drug_type, drug_barcode, drug_quantity),
            )
            db.commit()
        except mysql.connector.Error as err:
            db.rollback()
            print("Failed to add drug:", err)
        finally:
            cursor.close()

        return redirect(url_for("admin"))


@app.route("/edit_drug", methods=["POST"])
def edit_drug():
    if request.method == "POST":
        drug_name = request.form.get("drug_name")
        drug_type = request.form.get("drug_type")
        drug_barcode = request.form.get("drug_barcode")
        drug_quantity = request.form.get("drug_quantity")

        cursor = db.cursor()
        try:
            cursor.execute(
                """
                UPDATE DRUG
                SET NAME = %s, TYPE = %s, QUANTITY = %s
                WHERE BARCODE = %s
                """,
                (drug_name, drug_type, drug_quantity, drug_barcode),
            )
            db.commit()
        except mysql.connector.Error as err:
            db.rollback()
            print("Failed to update drug:", err)
        finally:
            cursor.close()

        return redirect(url_for("admin"))
```

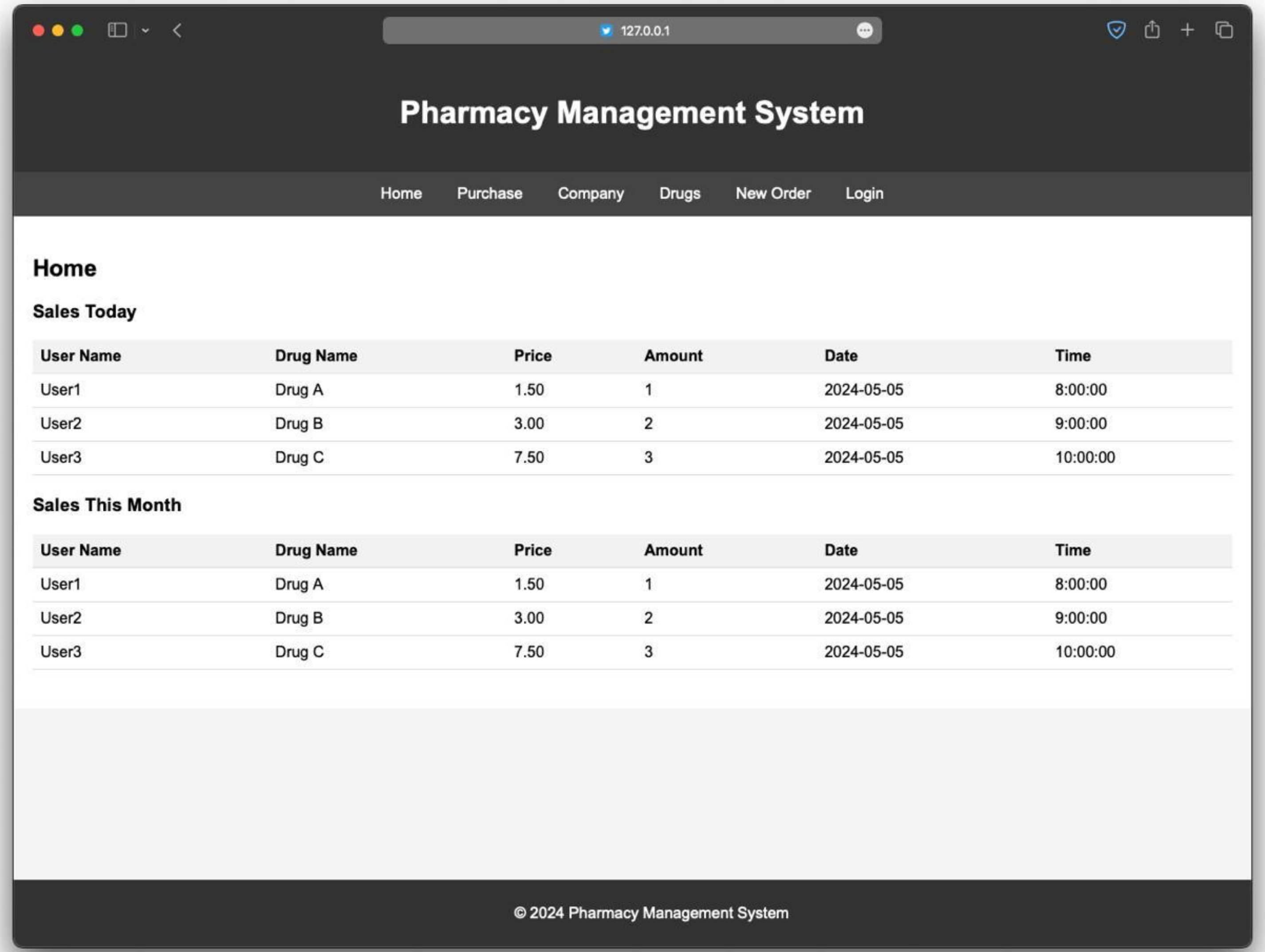
```
@app.route("/delete_drug/<drug_name>", methods=["DELETE"])
def delete_drug(drug_name):
    cursor = db.cursor()
    try:
        cursor.execute("DELETE FROM DRUG WHERE NAME = %s", (drug_name,))
        db.commit()
        return jsonify(success=True)
    except mysql.connector.Error as error:
        print("Failed to delete drug:", error)
        db.rollback()
        return jsonify(success=False, error="Failed to delete drug"), 500
    finally:
        cursor.close()


@app.route("/place_order", methods=["GET", "POST"])
def place_order():
    if request.method == "POST":
        drug_name = request.form.get("drug_name")
        dose = request.form.get("dose")
        drug_type = request.form.get("type")
        price = float(request.form.get("price"))
        amount = int(request.form.get("amount"))
        customer_name = request.form.get("customer_name")
        customer_phone = request.form.get("customer_phone")
        quantity = int(request.form.get("quantity"))

        cursor = db.cursor()
        try:
            cursor.execute(
                """
                INSERT INTO HISTORY_SALE (USER_NAME, BARCODE, DOSE, TYPE,
PRICE, AMOUNT, DATE, TIME, NAME, QUANTITY)
                VALUES (%s, %s, %s, %s, %s, %s, CURDATE(), CURTIME(), %s, %s)
                """,
                (
                    customer_name,
                    "123456",
                    dose,
                    drug_type,
                    price,
                    amount,
                    drug_name,
                    quantity,
                ),
            )
            db.commit()
        except mysql.connector.Error as err:
            db.rollback()
            print("Failed to place order:", err)
        finally:
            cursor.close()

        return redirect(url_for("index"))
    else:

        cursor = db.cursor()
        try:
            cursor.execute("SELECT NAME FROM DRUG")
            drugs = cursor.fetchall()
        except mysql.connector.Error as err:
            print("Failed to fetch drug names:", err)
            drugs = []
        finally:
            cursor.close()

        return render_template("order.html", drugs=drugs)


@app.route("/")
def index():
    sales_history_today = get_sales_history_today()
    sales_history_month = get_sales_history_month()
    sales_history = get_sales_history()
    purchases = get_purchases()
    companies = get_companies()
    drugs = get_drugs()

    return render_template(
        "index.html",
        sales_history=sales_history,
        purchases=purchases,
        companies=companies,
        drugs=drugs,
        sales_history_today=sales_history_today,
        sales_history_month=sales_history_month,
    )


if __name__ == "__main__":
    app.run(debug=True)
```

9 Result and Discussion:

The Pharmacy Management System GUI is designed to streamline the operations of a pharmacy by providing an intuitive and efficient interface for managing company information, drug inventory, purchase history, and user authentication. At its core, the GUI aims to enhance productivity and accuracy in day-to-day pharmacy tasks while ensuring compliance with regulatory standards. Users can easily navigate through different sections of the system, from accessing essential company details to updating drug information and processing new orders. The GUI prioritizes user-friendliness, with clear and organized layouts, intuitive navigation menus, and interactive features that facilitate smooth interactions. Whether it's managing inventory, tracking purchase history, or updating company information, the Pharmacy Management System GUI empowers users to effectively manage pharmacy operations while maintaining the highest standards of quality and compliance.

Additionally, the Pharmacy Management System GUI offers robust functionality to cater to the diverse needs of pharmacy staff, administrators, and customers alike. With features such as comprehensive search capabilities and detailed drug information displays, pharmacists can quickly find the medications they need and provide accurate information to patients. The system also includes tools for monitoring inventory levels, alerting staff to low stock, and facilitating seamless reordering to prevent stockouts and ensure continuity of care.

Furthermore, the GUI fosters secure and efficient communication between pharmacy personnel and suppliers, streamlining the procurement process and ensuring timely delivery of pharmaceutical supplies. Through the integration of user authentication and access control mechanisms, the system maintains data integrity and confidentiality, safeguarding sensitive information from unauthorized access. Overall, the Pharmacy Management System GUI serves as a centralized platform for managing all aspects of pharmacy operations, promoting efficiency, accuracy, and compliance while delivering exceptional service to patients and customers.


127.0.0.1

Pharmacy Management System

Home

Sales Today

User Name	Drug Name	Price	Amount	Date	Time
User1	Drug A	1.50	1	2024-05-05	8:00:00
User2	Drug B	3.00	2	2024-05-05	9:00:00
User3	Drug C	7.50	3	2024-05-05	10:00:00

Sales This Month

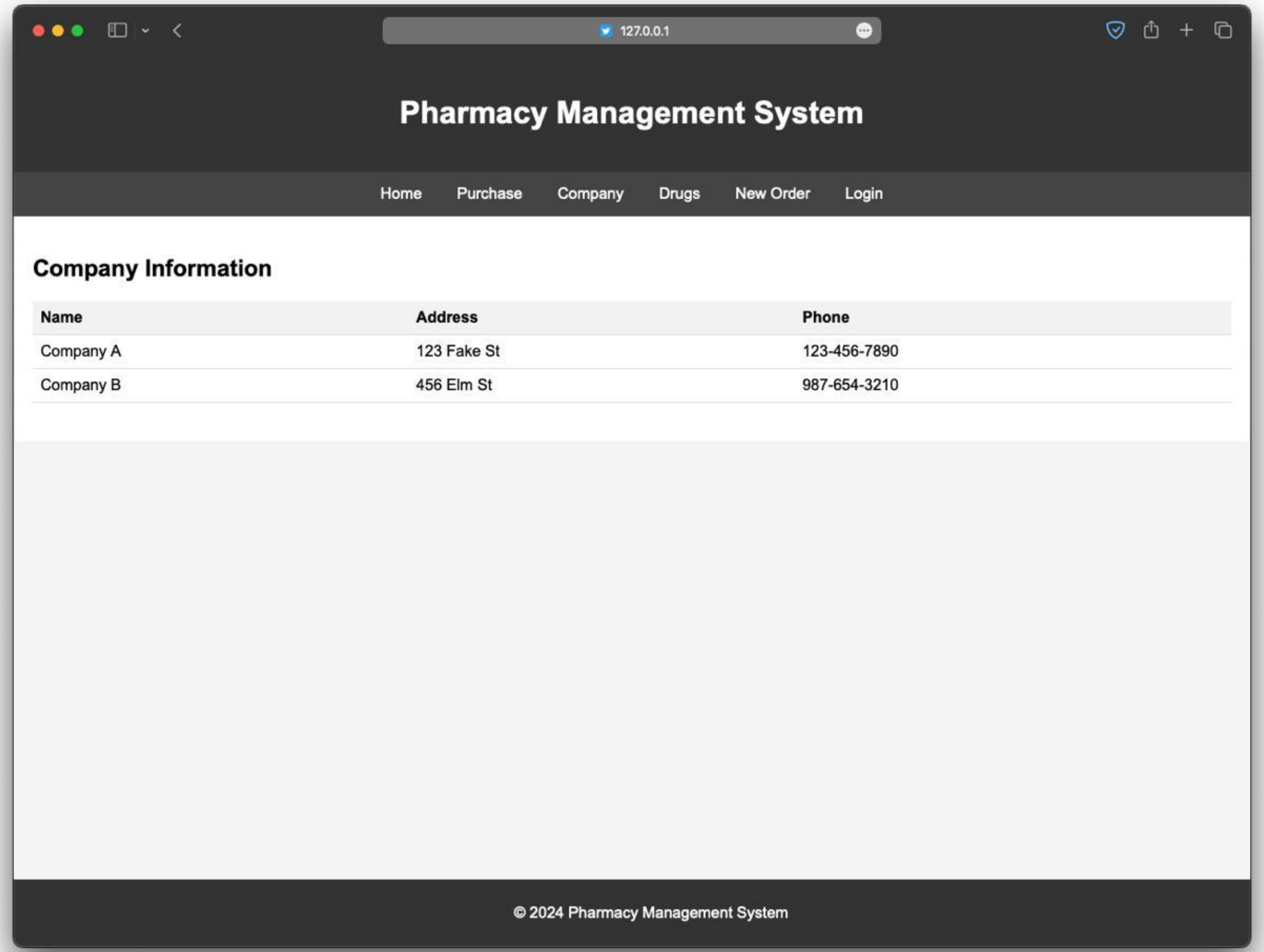
User Name	Drug Name	Price	Amount	Date	Time
User1	Drug A	1.50	1	2024-05-05	8:00:00
User2	Drug B	3.00	2	2024-05-05	9:00:00
User3	Drug C	7.50	3	2024-05-05	10:00:00

© 2024 Pharmacy Management System

Fig 9.1

Home Page

Welcome to the Pharmacy Management System. This page serves as the main hub for users to navigate through the system's functionalities. It typically includes options for accessing different sections of the system such as company information, drug information, purchase history, and login.



Company Information

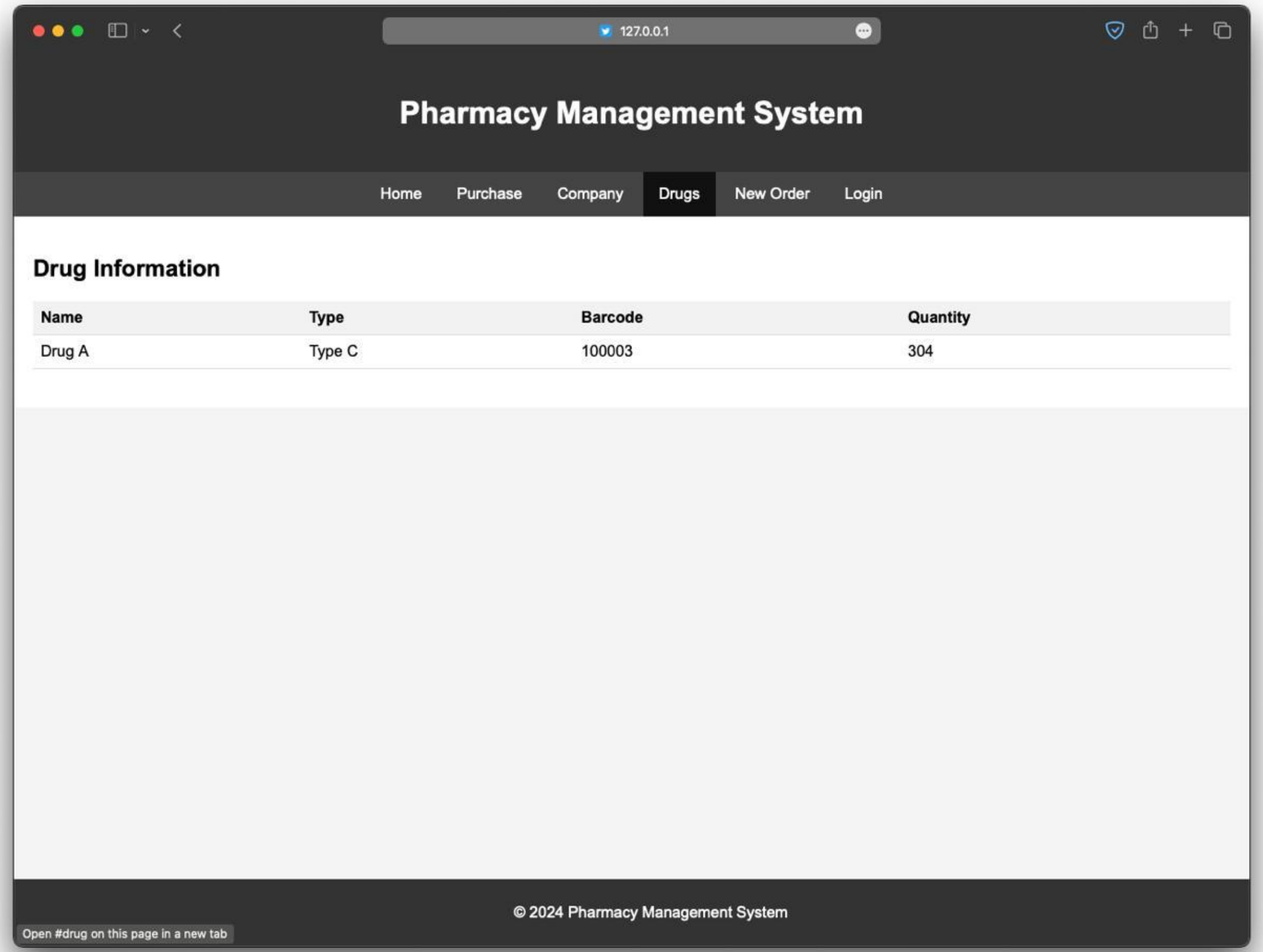
Name	Address	Phone
Company A	123 Fake St	123-456-7890
Company B	456 Elm St	987-654-3210

© 2024 Pharmacy Management System

Fig 9.2

Company Information Page

Company Information Page give the information about the company including address and phone number.
This page provides details about the pharmacy company such as its name, address, contact information, and any other relevant information. Users can view and edit company details from this page.



Pharmacy Management System

Drug Information

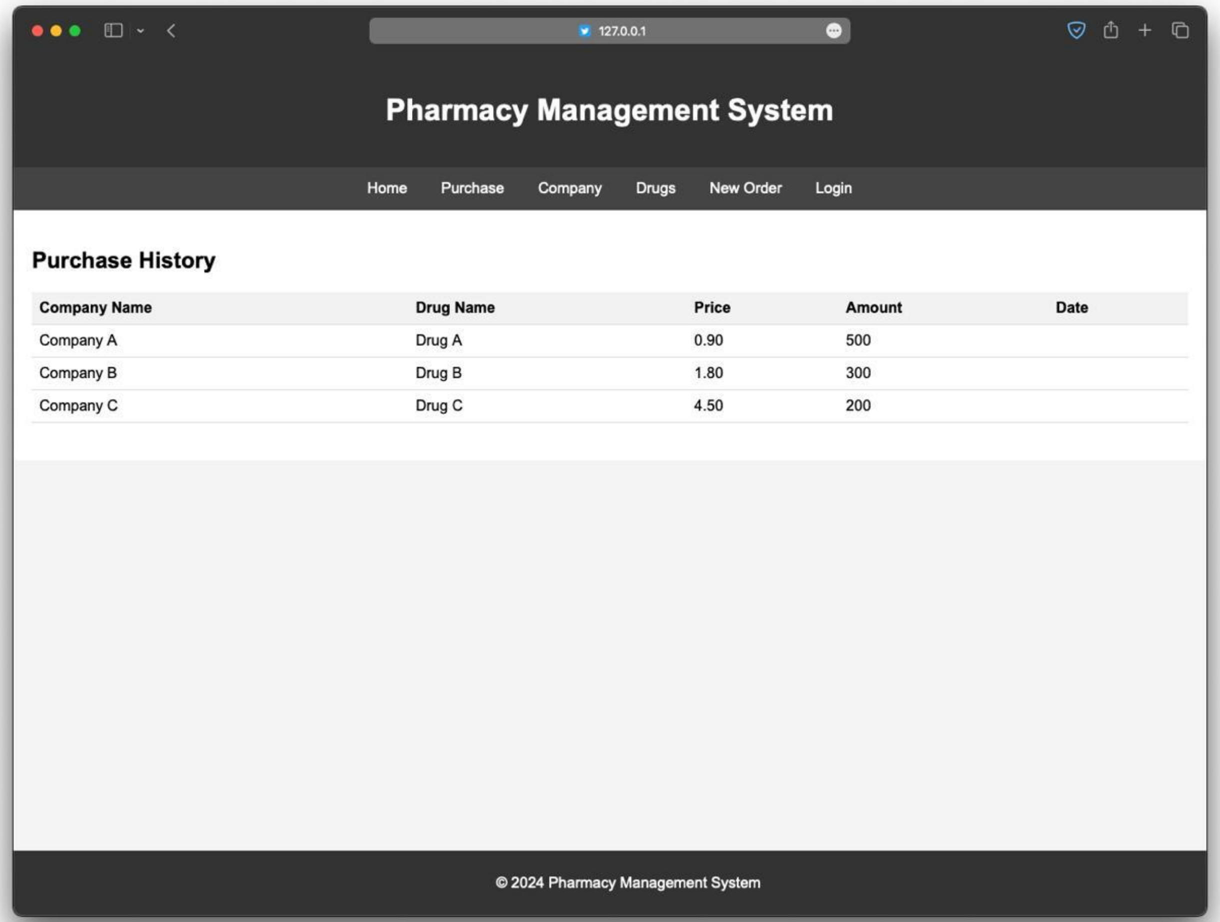
Name	Type	Barcode	Quantity
Drug A	Type C	100003	304

© 2024 Pharmacy Management System

Fig 9.3

Drug Information Page

Here, users can search for information about various drugs available in the pharmacy. It includes details like drug name, dosage, usage instructions, side effects, and availability. Users may also have the option to add new drugs or update existing drug information. Drug Information Page gives the information about Drugs, including its type, Barcode and Quantity.



127.0.0.1

Pharmacy Management System

Purchase History

Company Name	Drug Name	Price	Amount	Date
Company A	Drug A	0.90	500	
Company B	Drug B	1.80	300	
Company C	Drug C	4.50	200	

© 2024 Pharmacy Management System

Fig 9.4

Purchase History Page

Purchase History Page has all the information regarding the purchases made. This page displays a history of all purchases made by the pharmacy, including details like date of purchase, quantity, price, and supplier information. Users can search, filter, and view past purchase records for better inventory management and tracking.

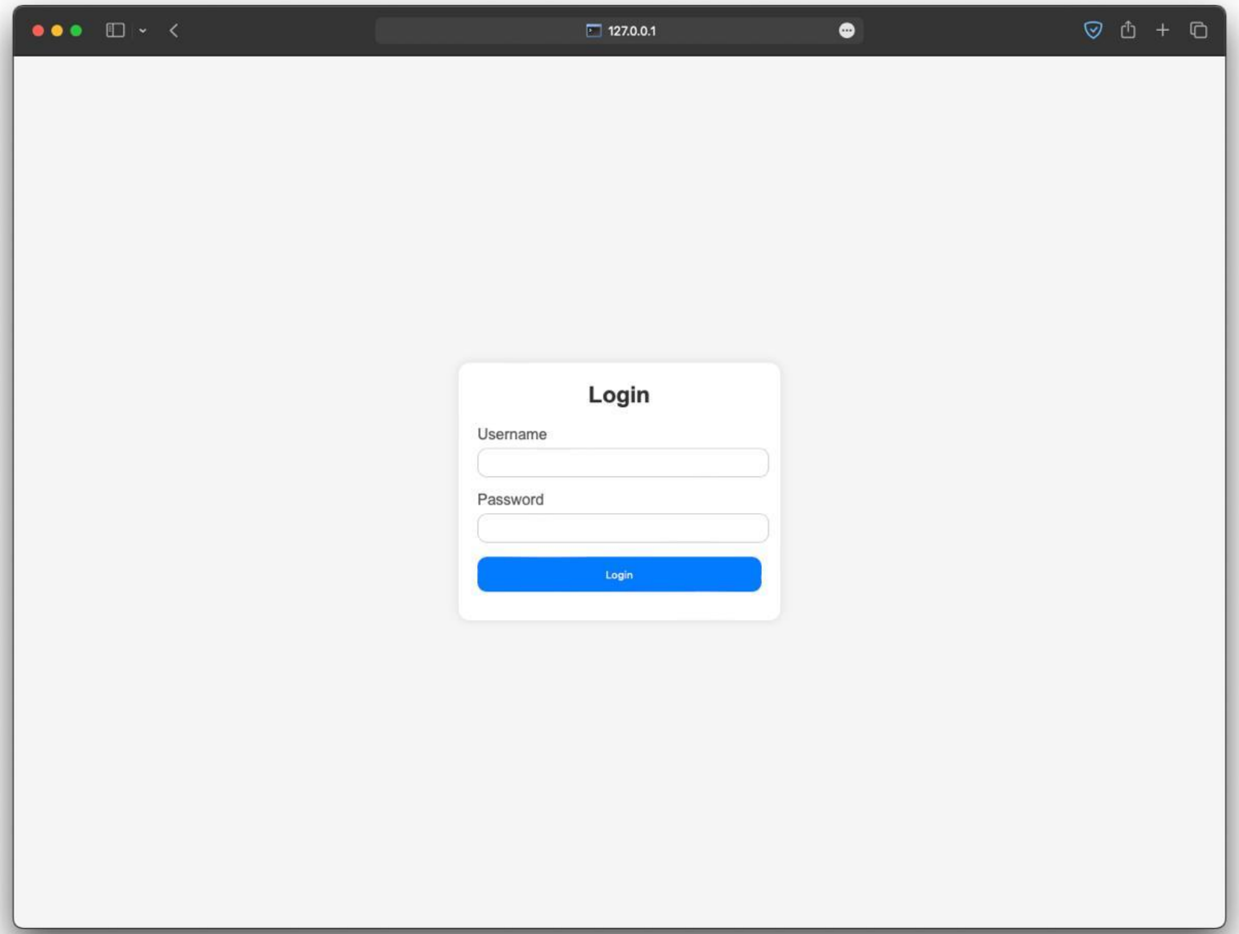
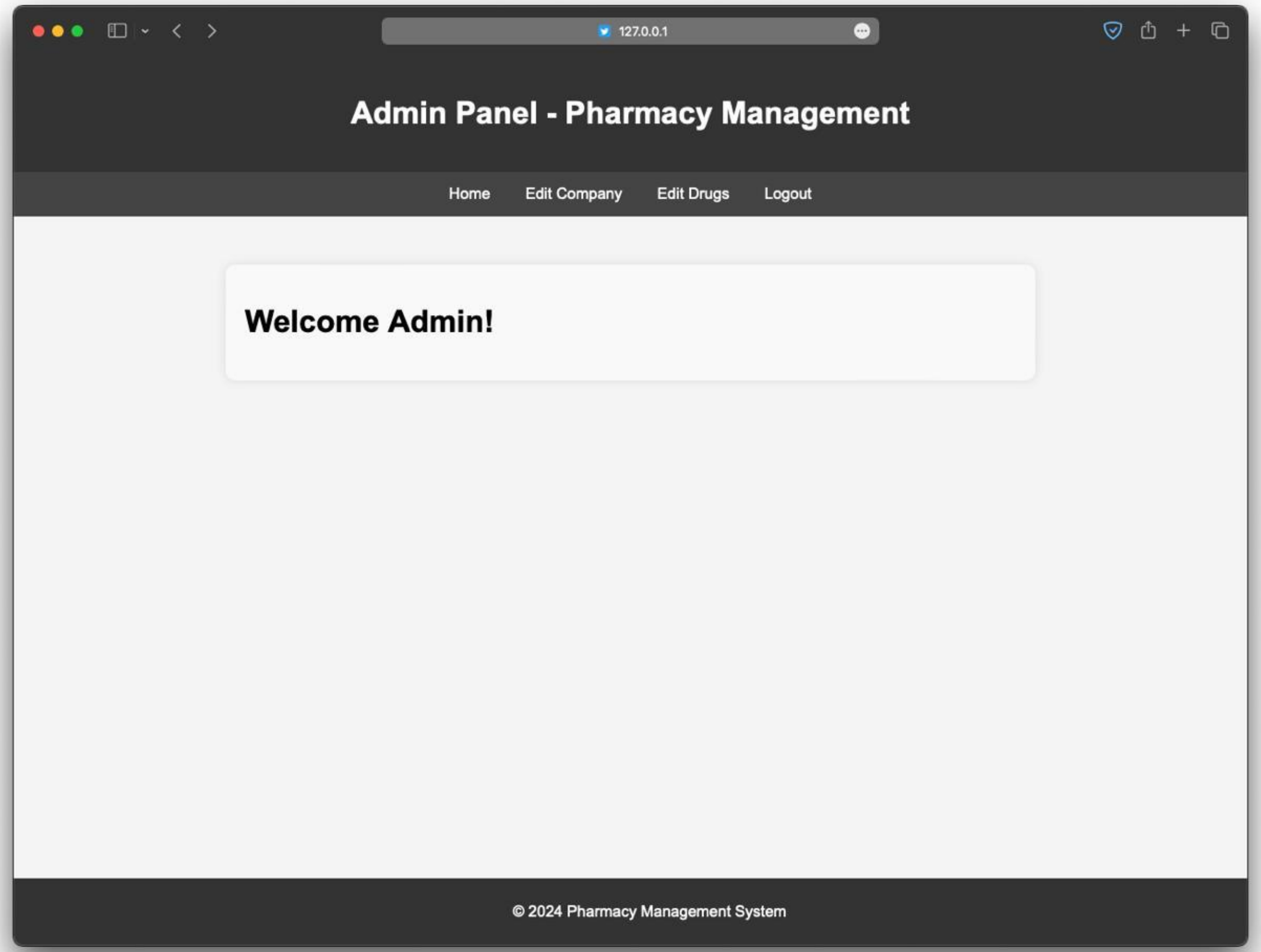


Fig 9.5

Login Page

This is the entry point for users to access the system. Users are required to provide valid credentials (such as username and password) to authenticate and gain access to the system's functionalities. This page may also include options for user registration and password recovery.

Login Page is the page from where the Admin can login and access the database.



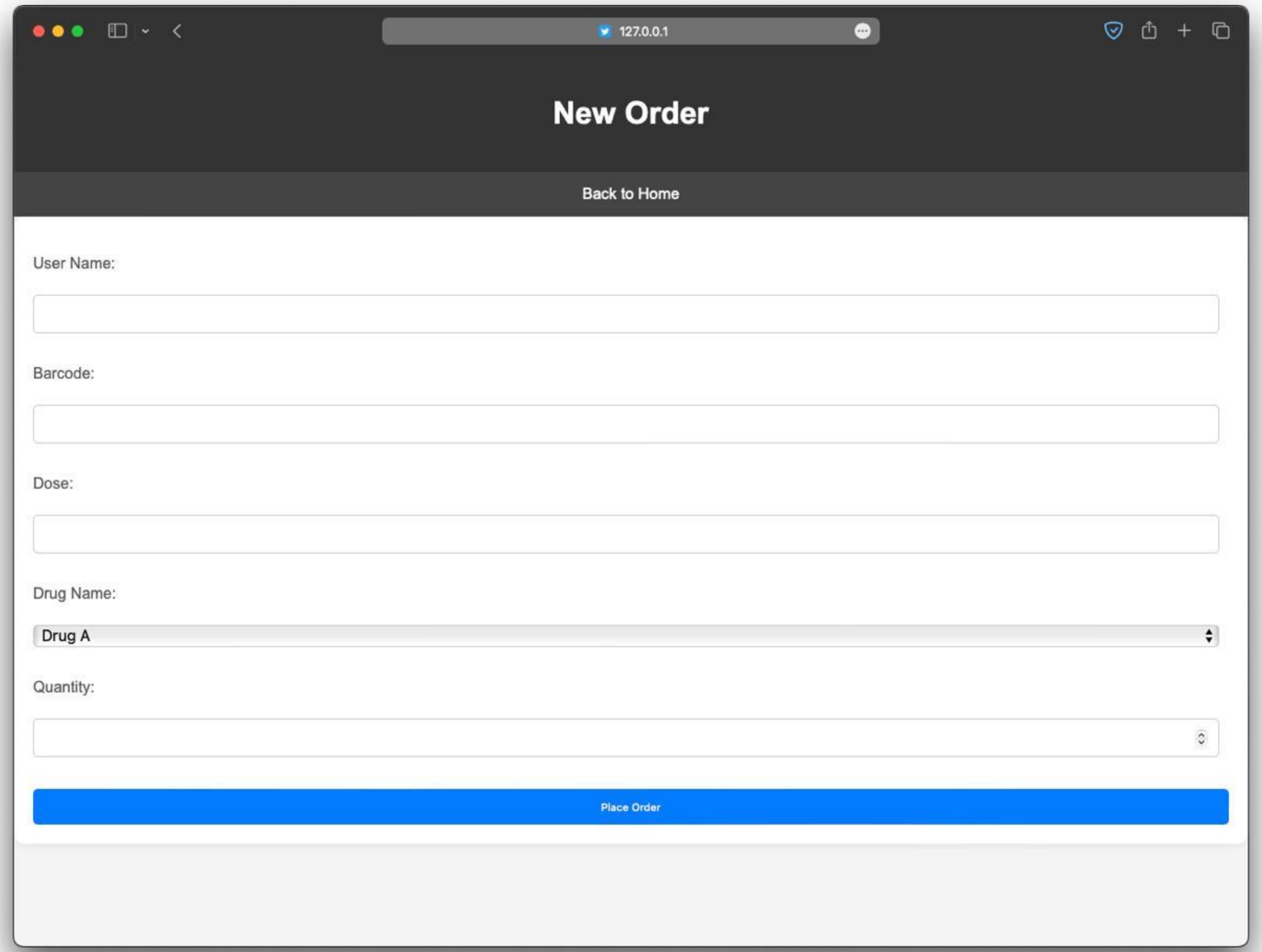
Welcome Admin!

© 2024 Pharmacy Management System

Fig 9.6

Welcome Admin Page

Upon successful login, administrators are greeted with a welcome page displaying relevant announcements, notifications, or system updates. It may also provide quick links to frequently used features or tasks specific to the admin role.



127.0.0.1

New Order

Back to Home

User Name:

Barcode:

Dose:

Drug Name:

Drug A

Quantity:

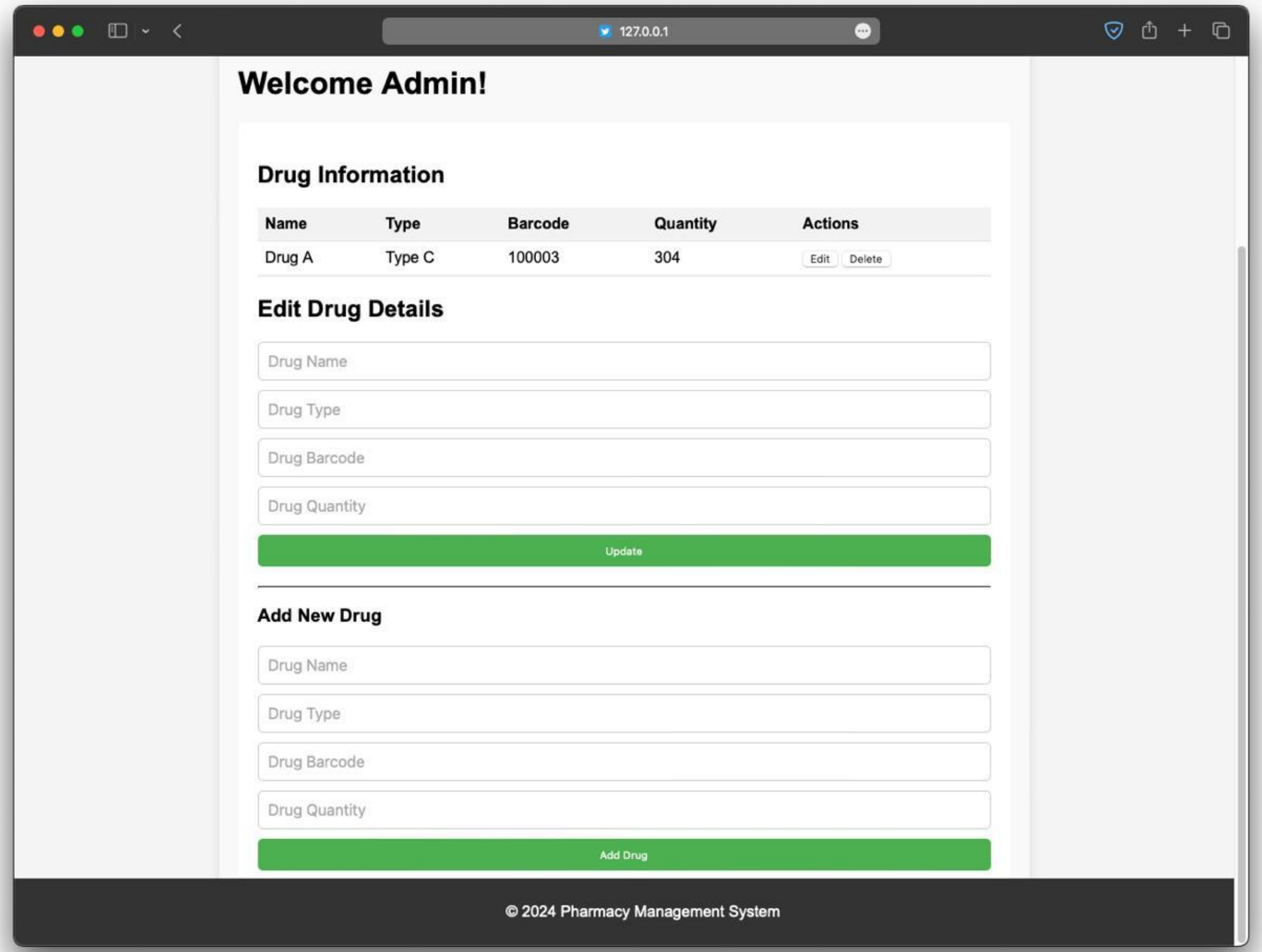
Place Order

Fig 9.7

New Order Page

New Order Page helps in dealing with customer. While selling the drugs, we will use New Order Page.

This page allows users to place new orders for drugs or pharmaceutical supplies. Users can specify the quantity, select the desired items from the available inventory, and provide shipping details. It may include features for order confirmation, tracking, and payment processing.



127.0.0.1

Welcome Admin!

Drug Information

Name	Type	Barcode	Quantity	Actions
Drug A	Type C	100003	304	Edit Delete

Edit Drug Details

Drug Name

Drug Type

Drug Barcode

Drug Quantity

Update

Add New Drug

Drug Name

Drug Type

Drug Barcode

Drug Quantity

Add Drug

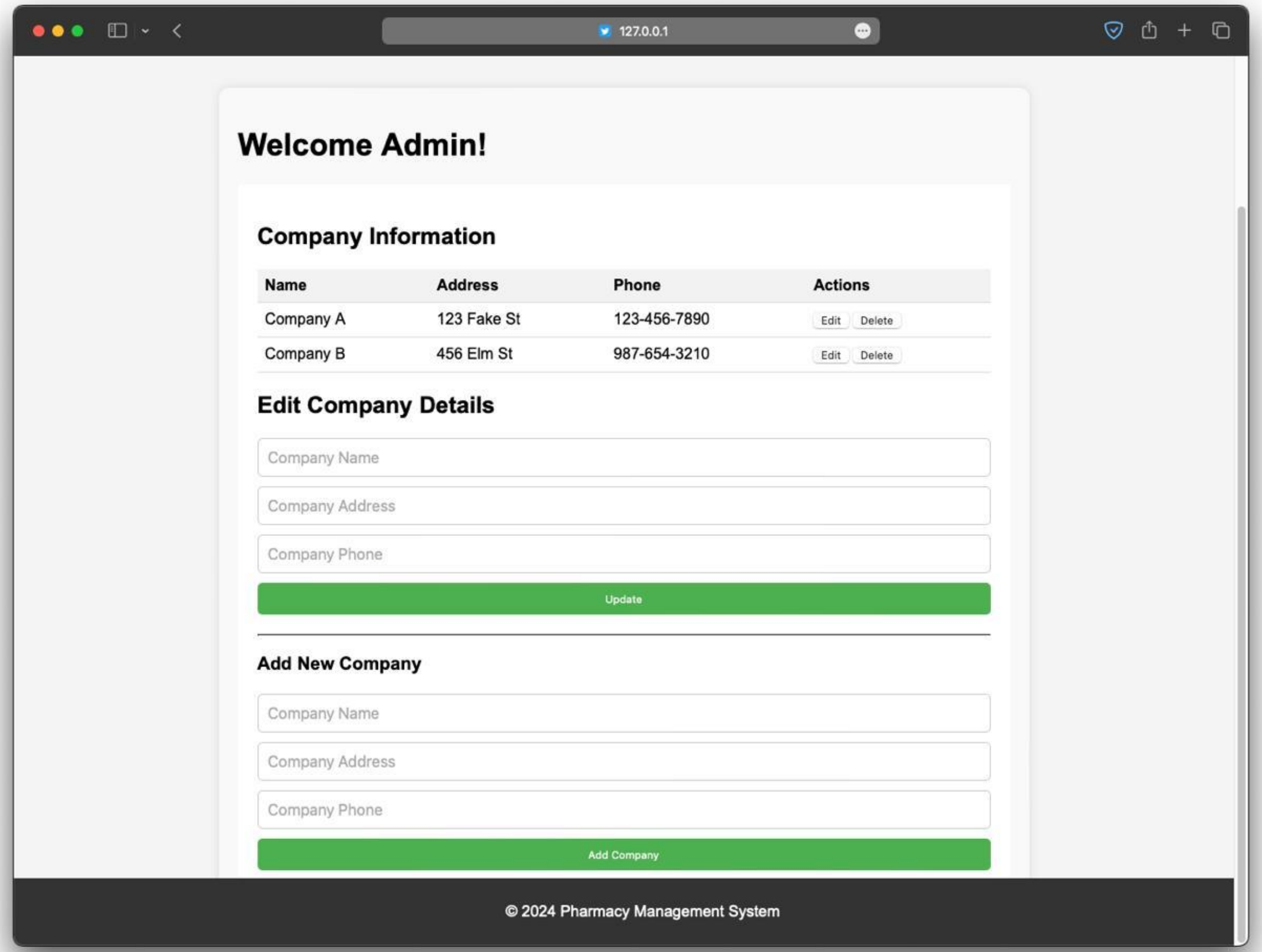
© 2024 Pharmacy Management System

Fig 9.8

Edit Drug Information Page

Edit Drug Information Page helps in editing the Drug Information by the Admin.

Users with appropriate permissions can access this page to modify or update existing drug information. They can edit details such as dosage, usage instructions, side effects, and pricing. Changes made here are reflected in the drug information database.



127.0.0.1

Welcome Admin!

Company Information

Name	Address	Phone	Actions
Company A	123 Fake St	123-456-7890	Edit Delete
Company B	456 Elm St	987-654-3210	Edit Delete

Edit Company Details

Company Name

Company Address

Company Phone

Update

Add New Company

Company Name

Company Address

Company Phone

Add Company

© 2024 Pharmacy Management System

Fig 9.9

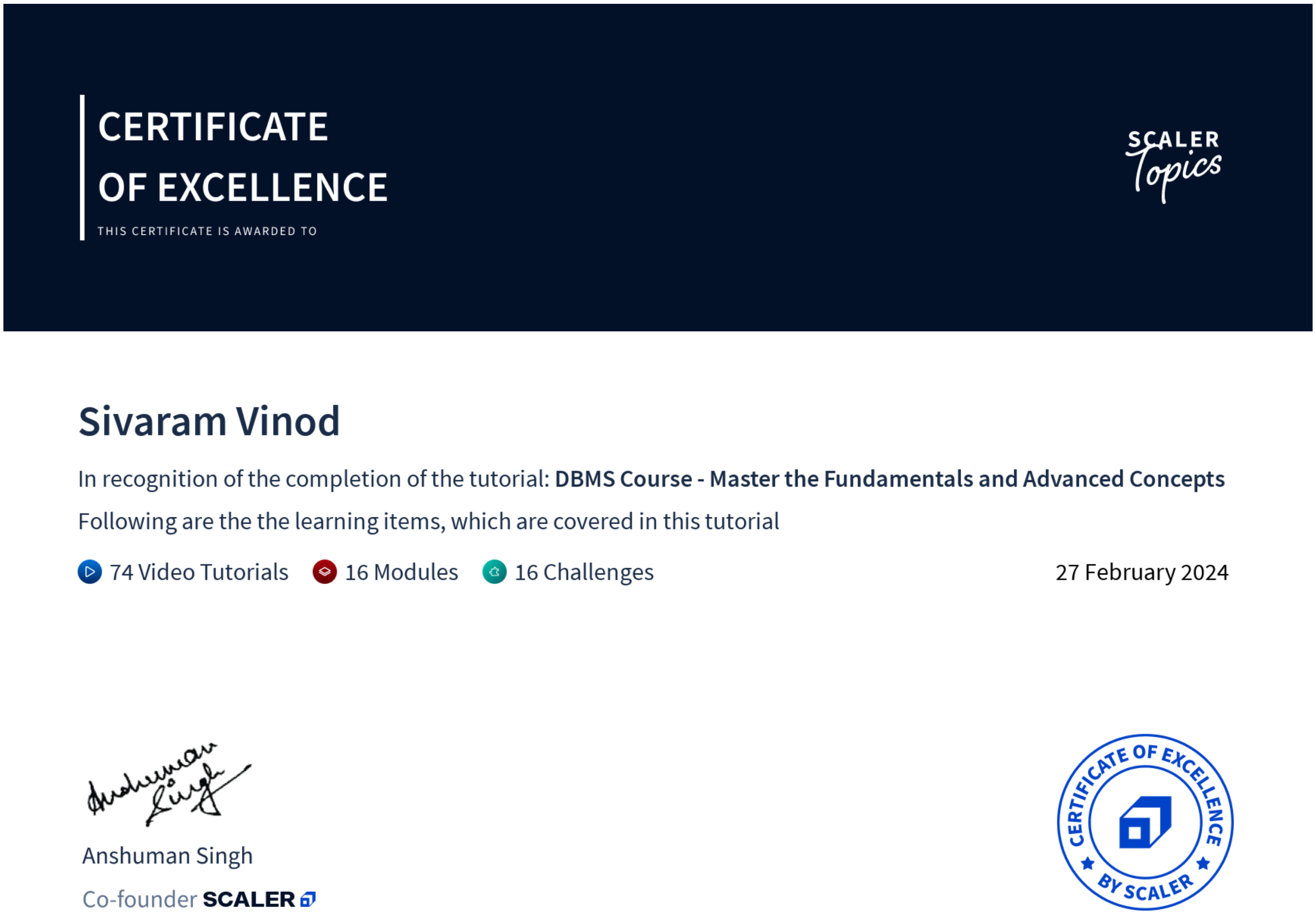
Edit Company Details Page

Edit Company Details Page helps the Admin to edit the details of the companies.

This page enables authorized users to edit and update company details such as name, address, contact information, and licensing information. It ensures that the pharmacy's information is accurate and up-to-date.

10 Online Certification:

SCALER DBMS Certificate of Sivaram Vinod (RA2211003011275)



CERTIFICATE

OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER Topics

Sivaram Vinod

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges 27 February 2024

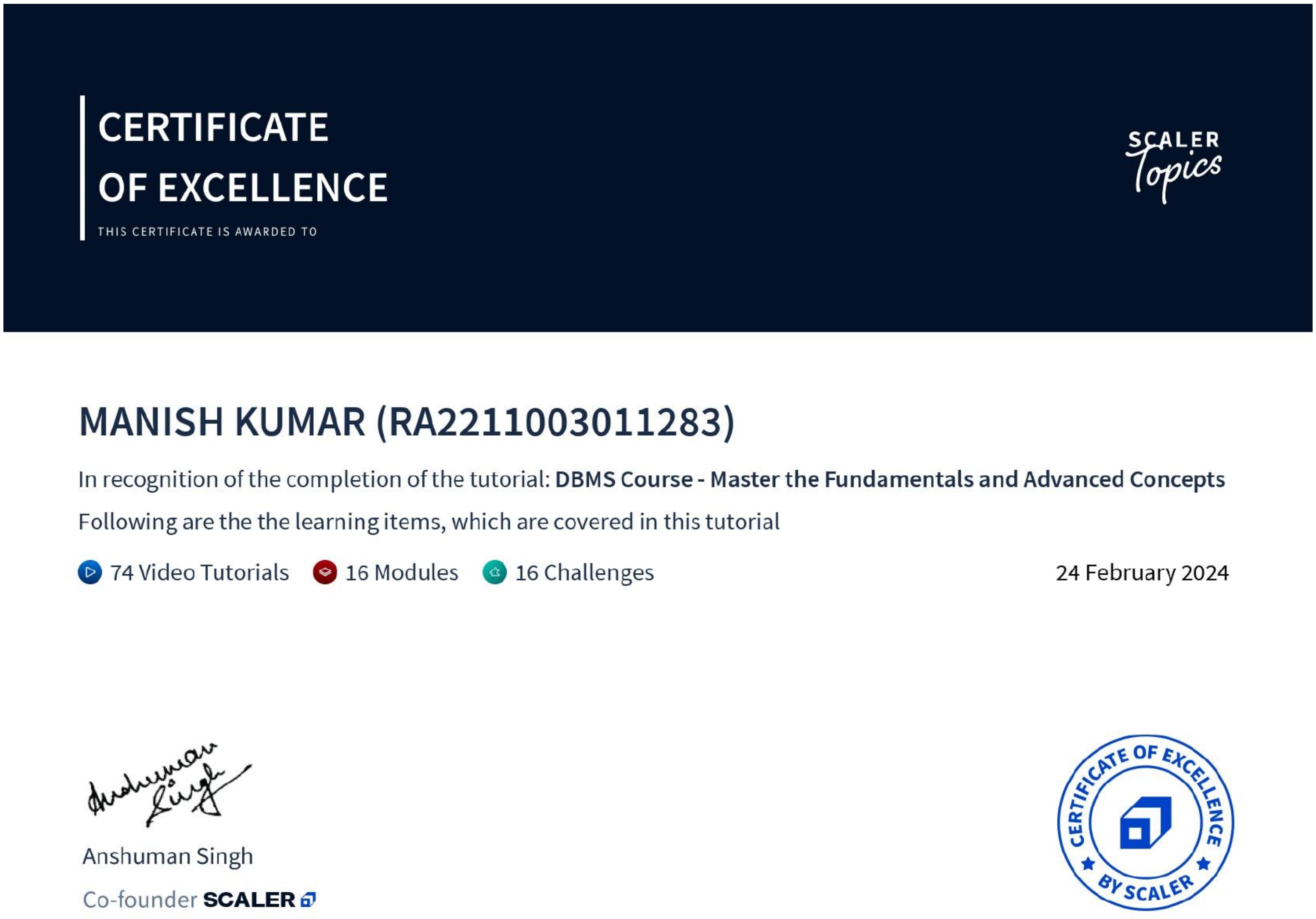
[illegible]

Anshuman Singh

Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE BY SCALER

SCALER DBMS Certificate of Manish Kumar (RA2211003011283)



CERTIFICATE
OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER Topics

MANISH KUMAR (RA2211003011283)

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges

24 February 2024

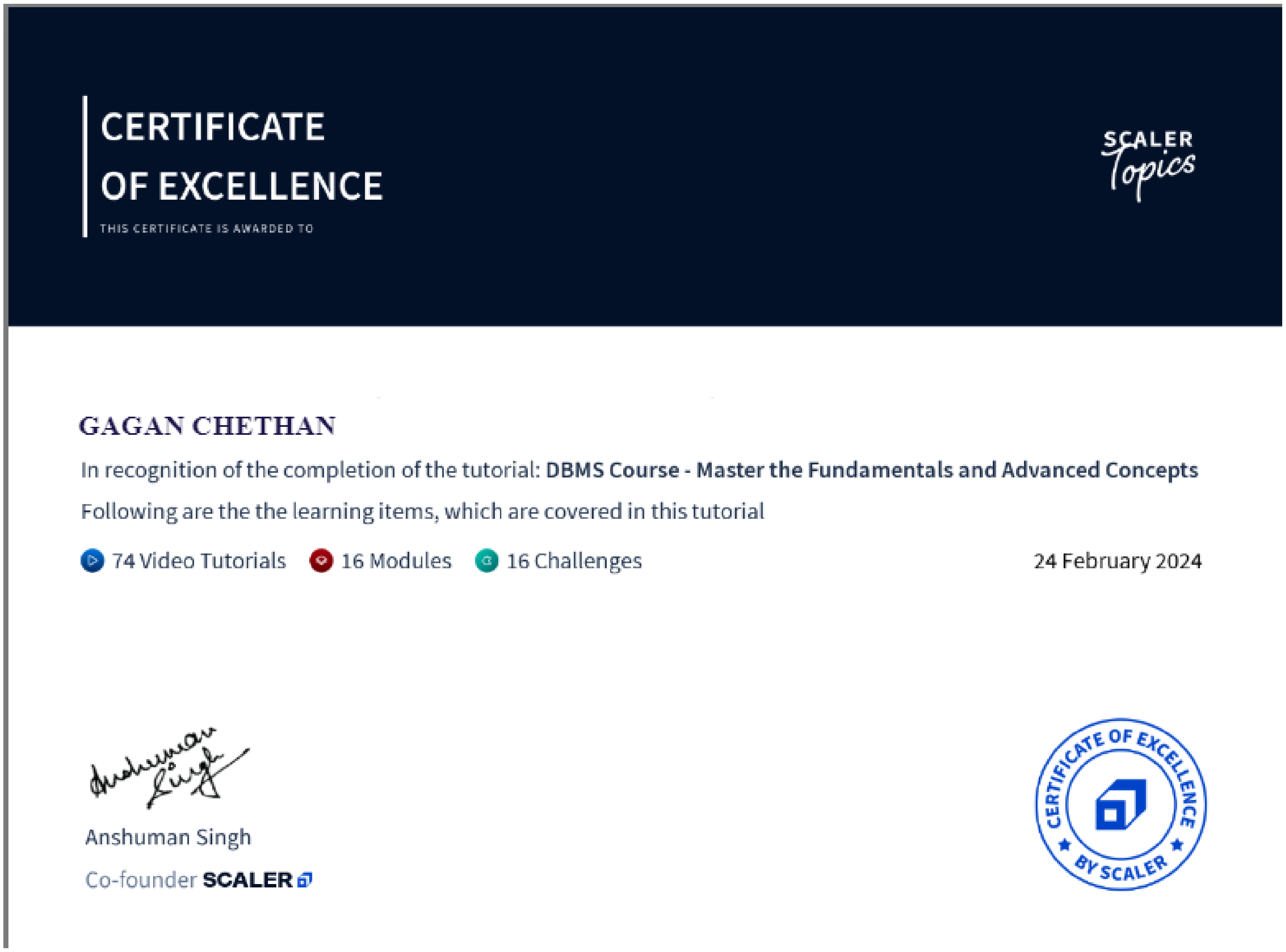
Anshuman Singh

Anshuman Singh

Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE BY SCALER

SCALER DBMS Certificate of Gagan Chethan (RA2211003011335)



CERTIFICATE

OF EXCELLENCE

THIS CERTIFICATE IS AWARDED TO

SCALER Topics

GAGAN CHETHAN

In recognition of the completion of the tutorial: **DBMS Course - Master the Fundamentals and Advanced Concepts**

Following are the the learning items, which are covered in this tutorial

74 Video Tutorials 16 Modules 16 Challenges

24 February 2024

Anshuman Singh

Co-founder **SCALER**

CERTIFICATE OF EXCELLENCE ★ BY SCALER ★

11 CONCLUSION

The implementation of a Pharmacy Management System (PMS) promises to revolutionize the operational efficiency and service quality within pharmaceutical establishments. By seamlessly integrating various functions such as inventory management, prescription processing, and customer records, PMS streamlines workflow, reduces errors, and enhances overall patient care. Automated processes not only alleviate the burden of manual tasks but also provide real-time insights into stock levels, medication usage patterns, and revenue streams, empowering pharmacists to make data-driven decisions for optimal resource allocation and business growth.

Moreover, the adoption of a PMS fosters compliance with regulatory standards, ensuring adherence to industry regulations and safeguarding patient confidentiality. With features like barcode scanning for accurate medication dispensing and built-in safeguards to prevent medication errors, PMS not only improves operational efficiency but also prioritizes patient safety. In essence, the integration of a Pharmacy Management System represents a pivotal step towards modernizing pharmaceutical practices, promoting transparency, and elevating the standard of care provided to patients while concurrently fortifying the sustainability and competitiveness of pharmacies in a rapidly evolving healthcare landscape.

REFERENCES

- **Wikipedia:** https://en.wikipedia.org/wiki/Pharmacy_management_system provides a general overview of pharmacy management systems, including their purpose, functionalities, and a brief history.

- **Software review websites:** Websites like Capterra (https://www.capterra.com/pharmacy-software/?page=3) list and review various pharmacy management software options. These can be helpful in understanding the features and benefits offered by different systems.

- **Software vendor websites:** Many companies develop pharmacy management software. Their websites will detail the specific features and functionalities of their systems.

- **Industry publications:** Publications geared towards pharmacists and pharmacy technicians may discuss pharmacy management systems and their impact on the profession.